**Production Scheduler**

Documentazione



**Docente:** Patrizia Scandurra

**Studenti:** Luca Fioretti, Stefano Sanvito

**Esame:** Informatica III (Progettazione e Algoritmi)

**Corso:** Ing. Informatica Magistrale

Indice

[1. Prefazione 5](#_Toc76022797)

[1.1 Versioni 5](#_Toc76022798)

[1.2 Documentazione Versioni 5](#_Toc76022799)

[2. Introduzione 6](#_Toc76022800)

[2.1 Teoria della schedulazione 6](#_Toc76022801)

[2.2 Schedulazione della produzione 6](#_Toc76022802)

[2.3 Scopo applicazione 6](#_Toc76022803)

[2.4 Toolchain 8](#_Toc76022804)

[3. Iterazione 0 9](#_Toc76022805)

[3.1 Architettura Hardware 9](#_Toc76022806)

[3.2 Architettura Software 9](#_Toc76022807)

[3.3 Specifiche 11](#_Toc76022808)

[**3.3.1 Funzionali** 11](#_Toc76022809)

[**3.3.2 Non funzionali** 12](#_Toc76022810)

[3.4 Casi d’uso 14](#_Toc76022811)

[**3.4.1 UC1 – Autenticazione** 14](#_Toc76022812)

[**3.4.2 UC2 – Visualizzazione Dati Utenti** 14](#_Toc76022813)

[**3.4.3 UC3 – Creazione/Modifica Dati Utenti** 14](#_Toc76022814)

[**3.4.4 UC4 – Visualizzazione Ruoli Utenti** 15](#_Toc76022815)

[**3.4.5 UC5 – Creazione/Modifica Ruoli Utenti** 15](#_Toc76022816)

[**3.4.6 UC6 – Eliminazione Ruolo Utente** 15](#_Toc76022817)

[**3.4.7 UC7 – Visualizzazione Prodotti** 16](#_Toc76022818)

[**3.4.8 UC8 – Creazione/Modifica Prodotto** 16](#_Toc76022819)

[**3.4.9 UC9 – Eliminazione Prodotto** 17](#_Toc76022820)

[**3.4.10 UC10 – Visualizzazione Tipi Macchina** 17](#_Toc76022821)

[**3.4.11 UC11 – Creazione/Modifica Tipo Macchina** 17](#_Toc76022822)

[**3.4.12 UC12 – Eliminazione Tipo Macchina** 18](#_Toc76022823)

[**3.4.13 UC13 – Visualizzazione Macchine** 18](#_Toc76022824)

[**3.4.14 UC14 – Creazione/Modifica Macchina** 18](#_Toc76022825)

[**3.4.15 UC15 – Eliminazione Macchina** 19](#_Toc76022826)

[**3.4.16 UC16 – Visualizzazione Ordini Produzione non chiusi** 19](#_Toc76022827)

[**3.4.17 UC17 – Creazione/Modifica Ordine Produzione** 19](#_Toc76022828)

[**3.4.18 UC18 – Eliminazione Ordine di Produzione** 20](#_Toc76022829)

[3.5 Modello Architetturale 22](#_Toc76022830)

[**3.5.1 Component Diagram** 22](#_Toc76022831)

[3.6 Programma di test 25](#_Toc76022832)

[**3.6.1 Utilizzo di Postman** 25](#_Toc76022833)

[4. Iterazione 1 26](#_Toc76022834)

[4.1 UML Class Diagram 28](#_Toc76022835)

[4.2 Progettazione Database 29](#_Toc76022836)

[**4.2.1 Modello E/R Database** 29](#_Toc76022837)

[4.3 Testing 30](#_Toc76022838)

[**4.3.1 Analisi statica** 30](#_Toc76022839)

[**4.3.2 Analisi Dinamica** 31](#_Toc76022840)

[5. Iterazione 2 33](#_Toc76022841)

[5.1 Specifiche funzionali 33](#_Toc76022842)

[5.2 Casi d’uso 34](#_Toc76022843)

[**5.2.1 UC19 – Visualizza dati Profilo** 34](#_Toc76022844)

[**5.2.2 UC20 – Modifica dati Profilo** 34](#_Toc76022845)

[**5.2.3 UC21 – Chiusura ordini di produzione aperti** 34](#_Toc76022846)

[**5.2.4 UC22 – Visualizzazione Ordini Produzione chiusi** 35](#_Toc76022847)

[**5.2.5 UC23 – Riapertura Ordine Produzione Chiuso** 35](#_Toc76022848)

[**5.2.6 UC24 – Creazione Pianificazione Produzione** 35](#_Toc76022849)

[**5.2.7 UC25 – Visualizzazione Pianificazione di Produzione** 36](#_Toc76022850)

[**5.2.8 UC26 – Elimina Pianificazione di Produzione** 36](#_Toc76022851)

[5.3 Modello Architetturale 38](#_Toc76022852)

[**5.3.1 Component Diagram** 38](#_Toc76022853)

[**5.3.2 UML Class Diagram** 39](#_Toc76022854)

[5.4 Progettazione Database 40](#_Toc76022855)

[**5.4.1 Modello E/R Database** 40](#_Toc76022856)

[5.5 Testing 41](#_Toc76022857)

[**5.5.1 Analisi Statica** 41](#_Toc76022858)

[**5.5.2 Analisi Dinamica** 41](#_Toc76022859)

[6. Iterazione 3 43](#_Toc76022860)

[6.1 Docker 43](#_Toc76022861)

[6.2 Implementazione Docker 44](#_Toc76022862)

[7. Algoritmi 46](#_Toc76022863)

[7.1 Obiettivo Production Planning 46](#_Toc76022864)

[7.2 Utilizzo tecnica Greedy 47](#_Toc76022865)

[7.3 Algoritmo Pianificazione Produzione 48](#_Toc76022866)

[**7.3.1 Analisi Complessità** 49](#_Toc76022867)

[**7.3.2 Implementazione Algoritmo Con Linguaggio PHP** 50](#_Toc76022868)

[8. Struttura Progetto 53](#_Toc76022869)

# **1. Prefazione**

## **1.1 Versioni**

1. Versione 0.0 – 20/04/2021
2. Versione 1.0 – 28/05/2021
3. Versione 2.0 – 25/06/2021
4. Versione 3.0 – 29/06/2021

## **1.2 Documentazione Versioni**

* Versione 0: Analisi strumenti utilizzati per lo sviluppo dell’applicazione. Studio dei requisiti funzionali e non funzionali, modellazione dei casi d’uso iniziali e della configurazione iniziale dell’architettura software e hardware.
* Versione 1.0: Sviluppo dell’iterazione 1, implementazione delle prime specifiche funzionali. Svolgimento test.
* Versione 2.0: Analisi nuovi requisiti funzionali, modellazione nuovi casi d’uso. Sviluppo dell’iterazione 2, implementazione dell’algoritmo di pianificazione della produzione. Aggiunta di ulteriori specifiche funzionali. Svolgimento test.
* Versione 3.0: Implementazione Docker. Revisione codice.

# **2. Introduzione**

## **2.1 Teoria della schedulazione**

La schedulazione è una forma di processo decisionale, che consiste nell’allocare risorse finite in modo tale che un dato obiettivo venga ottimizzato. Esempi tipici di risorse limitate sono le macchine di un’impresa manifatturiera, le ore di manodopera settimanali di un operaio o di una commessa di un negozio, le piste di atterraggio/decollo di un aeroporto. Esempi di obiettivi da ottimizzare invece sono il tempo per completare un progetto, il tempo di attesa di un aereo in volo o di un lotto da produrre.

La schedulazione ha il compito di sincronizzare e tempificare la sequenza delle operazioni. La schedulazione contiene cioè anche le informazioni temporali che specificano le tempistiche delle singole attività, la schedulazione include quindi il sequenziamento.

## **2.2** **Schedulazione della produzione**

La schedulazione della produzione è l’attività principale che un ufficio della programmazione deve compiere. Questo consente di riesaminare e riallineare i dati di produzione, cercando l’ottimizzazione riducendo errori ed inefficienze.

La schedulazione come descritto sopra è un processo decisionale che permette di creare e attribuire una sequenza di attività a ciascuna risorsa (attrezzature, macchine, operatori). Questa operazione tiene in considerazione i diversi vincoli tecnologici quali:

* Disponibilità
* Priorità
* Capacità produttiva

Gli obiettivi principali del processo di schedulazione sono:

* La minimizzazione dei costi di produzione
* Il raggiungimento dell’efficienza produttiva

## **2.3 Scopo applicazione**

Lo scopo dell’applicazione che si andrà a realizzare è quello di fornire ad una piccola azienda manifatturiera un sistema che possa aiutarla a pianificare la produzione dei propri ordini in modo da minimizzare il tempo di produzione degli stessi e ottimizzare l’efficienza dell’attività produttiva.

In particolare, gli operatori potranno gestire i dati relativi alle macchine, ai prodotti e agli ordini di produzione.

Vi sarà inoltre la possibilità di visualizzare gli ordini schedulati con i rispettivi tempi di completamento per consentire all’utente di verificare eventuali ritardi ed eventualmente trovare soluzioni per evitarli.

## **2.4 Toolchain**

Per lo sviluppo dell’applicazione è stata scelta la seguente toolchain:

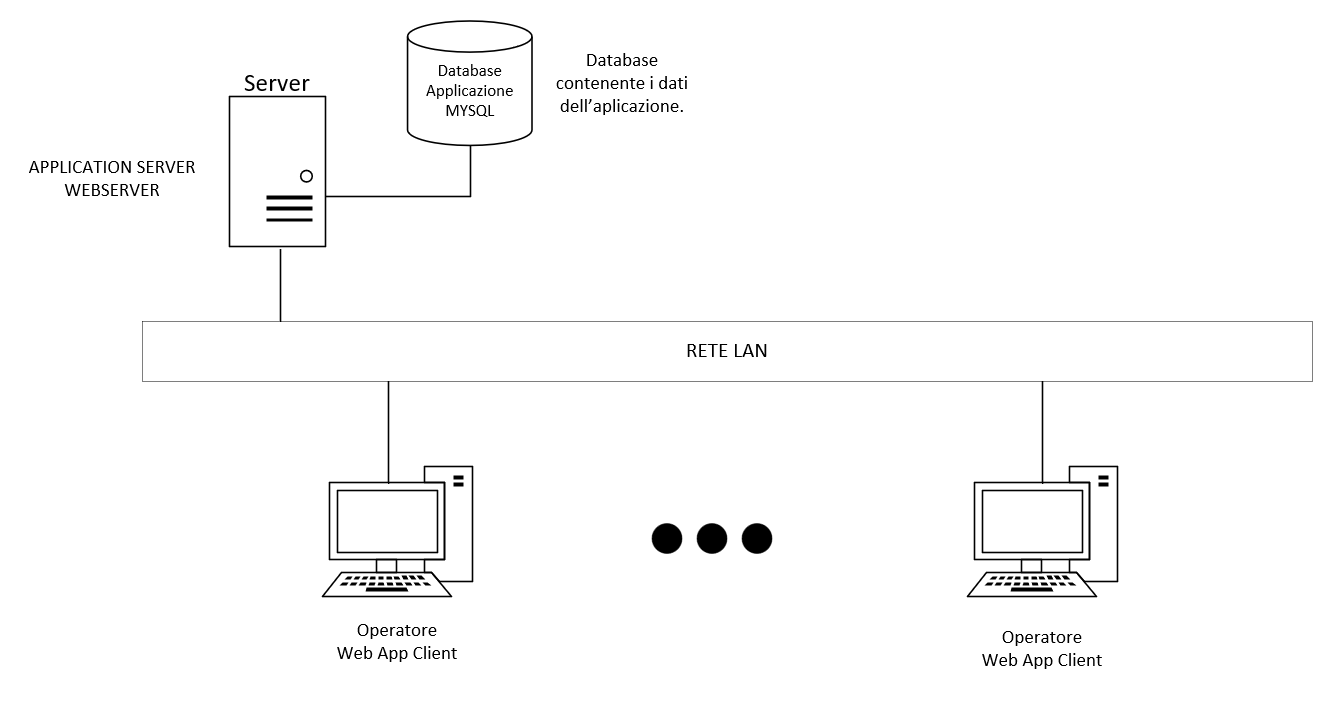
* **GitHub**, utilizzato per consentire il lavoro sul progetto a più persone. GitHub ha consentito l’utilizzo di branch secondari per lo sviluppo di particolari sezioni di codice più critiche, mantenendo sempre una versione funzionante dell’applicazione nel branch “master”.
* **GitHub Desktop**, applicazione che fornisce un’interfaccia grafica per l’utilizzo di GitHub.
* **Microsoft Visio**, utilizzato per la realizzazione di diagrammi, in particolare per la creazione del modello dell’architettura software, hardware e dei diagrammi UML.
* **PHPStorm**, utilizzato per lo sviluppo dell’applicazione. Scelto anche per la presenza di molti tool utili per velocizzare la scrittura di codice e la realizzazione di test. Implementa anche un tool per github che facilita l’utilizzo di tale piattaforma.
* **Laravel,** framework open source di tipo MVC scritto in PHP sul quale si baserà tutta l’applicazione.
* **Psalm**, tool per l’analisi statica del codice PHP
* **PHP Unit**, testing framework per la realizzazione dei test.
* **Postman,** utilizzato per testare le API in fase di sviluppo del backend dell’applicazione senza dover obbligatoriamente avere a disposizione il frontend.
* **Dbeaver,** utilizzato per la gestione del database.

# **3. Iterazione 0**

## **3.1 Architettura Hardware**

L’applicazione avrà una struttura client/server. Di conseguenza per poter utilizzare l’applicazione sarà necessaria la presenza di un server centrale per la fruizione di tutte le funzionalità del software. Il server dovrà essere accompagnato da un database per il salvataggio dei dati.

Di seguito l’architettura hardware necessaria per l’utilizzo dell’applicazione.



## **3.2 Architettura Software**

L’applicazione sarà progettata e realizzata tramite il framework Laravel; un framework open source di tipo MVC (Model View Controller) scritto in PHP.

L’utilizzo del framework Laravel impone una struttura di tipo MVC (Model View Controller).

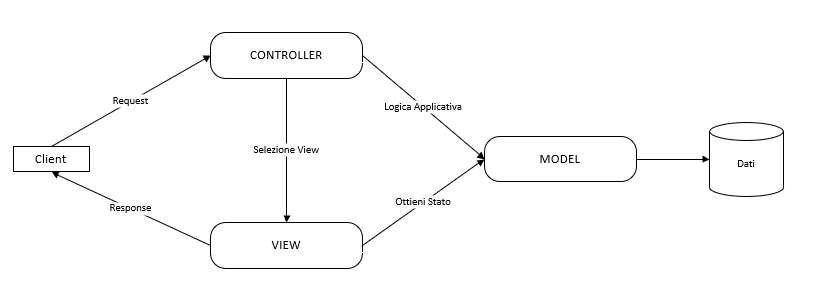
I componenti principali di questa architettura sono:

* **Model**, contiene i metodi di accesso ai dati, è il componente che si interfaccia direttamente con il database per recuperare, creare e modificare i dati.
* **View**, è il componente che si occupa di visualizzare i dati all’utente e gestisce l’interazione fra quest’ultimo e l’infrastruttura sottostante.
* **Controller,** è il componente posto a metà tra il Model e la View. Contiene metodi pubblici chiamati **action (API)** contenenti operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato nella View.

Nell’implementazione PHP questi metodi e i loro parametri vengono messi in correlazione con gli URL delle richieste http provenienti dal componente View.

Vantaggi utilizzo pattern MVC:

* Divide et impera: le tre tipologie di componenti possono essere progettati separatamente. Questo aspetto è utilissimo in fase di sviluppo perché, ad esempio, permette a più sviluppatori di agire contemporaneamente e indipendentemente gli uni dagli altri su aspetti del frontend e su aspetti del backend.
* I canali di comunicazione tra i tre componenti sono minimi.
* Solitamente le View e i Controller fanno uso di componenti riutilizzabili per vari tipi di controlli dell’interfaccia utente.
* E’ facile modificare uno dei tre componenti senza dover adeguare gli altri due.



Linguaggi utilizzati:

* Back-end: PHP
* Front-end: HTML, CSS, JavaScript

## **3.3 Specifiche**

### **3.3.1 Funzionali**

Le specifiche funzionali ricavate da questa prima analisi sono:

|  |
| --- |
| FU1 – Gestione Autenticazione |
| * Tutte le funzionalità dell’applicazione dovranno essere utilizzate solo da utenti registrati ed autenticati. * L’utente ha la possibilità di autenticarsi all’interno dell’applicazione. * L’utente ha la possibilità di eseguire il logout dall’applicazione. |

|  |
| --- |
| FU2 – Gestione Dati Utenti |
| * Gli utenti con il ruolo di amministratore possono eseguire le seguenti operazioni sui dati degli utenti: * Visualizzare i dati degli utenti * Creare un utente * Modificare gli utenti |

|  |
| --- |
| FU3 – Gestione Ruoli |
| * Gli utenti con il ruolo di amministratore possono eseguire le seguenti operazioni sui ruoli assegnabili agli utenti: * Visualizzare i ruoli * Creare un nuovo ruolo * Modificare i ruoli * Eliminare i ruoli * Assegnare i ruoli agli utenti |

|  |
| --- |
| FU4 – Gestione Macchine Produzione |
| * Gli utenti possono eseguire le seguenti operazioni sulle macchine di produzione: * Visualizzare le macchine * Creare una nuova macchina * Modificare le macchine * Eliminare le macchine |

|  |
| --- |
| FU5 – Gestione Tipologia Macchine |
| * Gli utenti possono eseguire le seguenti operazioni sulle tipologie delle macchine di produzione: * Visualizzare le tipologie * Creare una nuova tipologia di macchina * Modificare le tipologie delle macchine * Eliminare le tipologie delle macchine |

|  |
| --- |
| FU6 – Gestione Prodotti |
| * Gli utenti possono eseguire le seguenti operazioni sui prodotti: * Visualizzare i prodotti * Creare un nuovo prodotto * Modificare i prodotti * Eliminare i prodotti |

|  |
| --- |
| FU7 – Gestione Ordini di Produzione |
| * Gli utenti possono eseguire le seguenti operazioni sugli ordini di produzione: * Visualizzare gli ordini di produzione * Creare un nuovo ordine di produzione * Modificare gli ordini di produzione * Eliminare gli ordini di produzione |

L’applicazione deve inoltre fornire le seguenti funzionalità:

* Fornire tutte le operazioni di lettura/scrittura dei dati sul database.
* Fornire tutte le operazioni di manipolazione dei dati inseriti dall’utente per poterli adattare alle esigenze del database.
* Verificare le seguenti condizioni in fase di salvataggio dei dati sul database:
  1. Una tipologia di macchina di produzione può essere eliminata se e solo se non è già assegnata ad una macchina e/o ad un prodotto.
  2. Un ordine di produzione non può essere modificato e/o eliminato quando si trova in fase di produzione o quando la sua produzione è già stata ultimata.
  3. Il tempo di produzione di un prodotto non può essere modificato quando è contenuto in un ordine in fase di produzione.
  4. Un prodotto può essere eliminato se e solo se non è assegnato ad un ordine di produzione.
* Comunicare l’esito di tutte le operazioni eseguite dall’utente.

### **3.3.2 Non funzionali**

* **Usabilità:** l’applicazione deve essere semplice ed intuitiva da utilizzare. Si è cercato di mantenere lo stesso stile grafico in tutte le pagine in modo da abituare l’utente all’utilizzo della stessa. Lo stile inoltre è semplice e schematico in modo da facilitare l’utente nella consultazione delle informazioni e nell’utilizzo delle funzioni messe a disposizione dall’applicazione.

Per quasi la totalità delle azioni svolte dall’utente l’applicazione lo informa tramite un messaggio a schermo circa il risultato delle operazioni ed eventuali errori.

* **Efficienza:** la maggior parte delle operazioni richiedono tempi molto brevi per essere eseguite. L’unica operazione che potrebbe risultare più critica in questo senso è la creazione della pianificazione di produzione. Per questa operazione si è cercato infatti di trovare un algoritmo il più efficiente possibile in modo da minimizzare i tempi di attesa da parte dell’utilizzatore.
* **Portabilità:** in questa prima fase l’applicazione non ha particolari esigenze di portabilità, nelle fasi successive si potrebbe pensare ad utilizzare un sistema come quello fornito da Docker per aumentare la facilità di deployment dell’applicazione su piattaforme diverse.

## **3.4 Casi d’uso**

### **3.4.1 UC1 – Autenticazione**

* **Descrizione**: autenticazione utente nell’applicazione.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: presenza dell’utente nel database dell’applicazione.
* **Post** **condizioni**: l’utente ha accesso alle funzioni dell’applicazione.
* **Procedimento**:

1. L’utente raggiunge la pagina di login dell’applicazione.
2. L’utente inserisci le sue credenziali nel form di login.
3. L’applicazione controlla se i dati dell’utente sono corretti.
4. L’utente viene reindirizzato alla homepage dell’applicazione.

* **Eccezioni**:
  1. Viene visualizzato l’errore prodotto (es. credenziali errate).

### **3.4.2 UC2 – Visualizzazione Dati Utenti**

* **Descrizione**: visualizzazione in tabella degli utenti registrati nell’applicazione.
* **Attori** **coinvolti**: utente amministratore.
* **Precondizioni**: l’utente deve avere il ruolo di amministratore.
* **Post** **condizioni**: i dati degli utenti registrati vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione degli utenti.
  2. L’applicazione controlla che l’utente sia un amministratore.
  3. Vengono letti i dati relativi agli utenti dal db.
  4. Vengono mostrati a schermo i dati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’utente non abbia il permesso di visualizzare i dati degli utenti.

### **3.4.3 UC3 – Creazione/Modifica Dati Utenti**

* **Descrizione**: creazione/modifica degli utenti registrati nell’applicazione.
* **Attori** **coinvolti**: utente amministratore.
* **Precondizioni**: l’utente deve avere il ruolo di amministratore.
* **Post** **condizioni**: l’utente creato viene inserito nel database. I dati dell’utente modificato vengono aggiornati nel database
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi all’utente che si vuole creare/modificare.
  2. L’applicazione controlla che l’utente sia un amministratore.
  3. Vengono inseriti/modificati i dati del nuovo utente/utente da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’utente non abbia il permesso di visualizzare i dati degli utenti.

### **3.4.4 UC4 – Visualizzazione Ruoli Utenti**

* **Descrizione**: visualizzazione in tabella dei ruoli utente.
* **Attori** **coinvolti**: utente amministratore.
* **Precondizioni**: l’utente deve avere il ruolo di amministratore.
* **Post** **condizioni**: i dati dei ruoli vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione dei ruoli.
  2. L’applicazione controlla che l’utente sia un amministratore.
  3. Vengono letti i dati relativi ai ruoli dal db.
  4. Vengono mostrati a schermo i dati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’utente non abbia il permesso di visualizzare i dati degli utenti.

### **3.4.5 UC5 – Creazione/Modifica Ruoli Utenti**

* **Descrizione**: creazione/modifica dei ruoli utente.
* **Attori** **coinvolti**: utente amministratore.
* **Precondizioni**: l’utente deve avere il ruolo di amministratore.
* **Post** **condizioni**: il ruolo creato viene inserito nel database. I dati del ruolo modificato vengono aggiornati nel database
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi al ruolo che si vuole creare/modificare.
  2. L’applicazione controlla che l’utente sia un amministratore.
  3. Vengono inseriti/modificati i dati del nuovo ruolo/ruolo da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’utente non abbia il permesso di visualizzare i dati degli utenti.

### **3.4.6 UC6 – Eliminazione Ruolo Utente**

* **Descrizione**: eliminazione ruolo utente.
* **Attori** **coinvolti**: utente amministratore.
* **Precondizioni**: l’utente deve avere il ruolo di amministratore. Il ruolo che si vuole cancellare non deve essere assegnato a nessun utente registrato nell’applicazione.
* **Post** **condizioni**: il ruolo viene eliminato dal database.
* **Procedimento**:
  1. L’utente seleziona il ruolo che vuole eliminare.
  2. L’applicazione controlla che l’utente sia un amministratore e che il ruolo da cancellare non sia assegnato a nessun utente registrato nell’applicazione.
  3. Viene eliminato il ruolo dal db.
  4. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’utente non abbia il permesso di visualizzare i dati degli utenti.
  2. Viene visualizzato un messaggio di errore qualora il ruolo non possa essere eliminato per i motivi sopra citati.

### **3.4.7 UC7 – Visualizzazione Prodotti**

* **Descrizione**: visualizzazione in tabella dei prodotti.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: i dati dei prodotti vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione dei dati dei prodotti.
  2. Vengono letti dal database i dati relativi ai prodotti.
  3. Vengono mostrati a schermo i dati relativi ai prodotti tramite una tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina di visualizzazione dei prodotti non sia disponibile.

### **3.4.8 UC8 – Creazione/Modifica Prodotto**

* **Descrizione**: creazione/modifica prodotto.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: il prodotto creato viene inserito nel database. I dati del prodotto modificato vengono aggiornati nel database.
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi al prodotto che si vuole creare/modificare.
  2. L’applicazione controlla che i dati inseriti dall’utente siano coerenti.
  3. Vengono inseriti/modificati i dati del nuovo ruolo/ruolo da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di creazione/modifica non sia andata a buon fine.

### **3.4.9 UC9 – Eliminazione Prodotto**

* **Descrizione**: eliminazione prodotto.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. Il prodotto che si vuole cancellare non deve essere assegnato a nessun ordine di produzione.
* **Post** **condizioni**: il prodotto viene eliminato dal database.
* **Procedimento**:
  1. L’utente seleziona il prodotto che vuole eliminare.
  2. L’applicazione controlla che il prodotto da cancellare non sia assegnato a nessun ordine di produzione.
  3. Viene eliminato il prodotto dal db.
  4. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora il prodotto non possa essere eliminato per i motivi sopra citati.

### **3.4.10 UC10 – Visualizzazione Tipi Macchina**

* **Descrizione**: visualizzazione in tabella dei tipi macchina.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: i dati dei tipi macchina vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione dei dati dei tipi macchina.
  2. Vengono letti dal database i dati relativi ai tipi macchina.
  3. Vengono mostrati a schermo i dati relativi ai tipi macchina tramite una tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina di visualizzazione dei tipi macchina non sia disponibile.

### **3.4.11 UC11 – Creazione/Modifica Tipo Macchina**

* **Descrizione**: creazione/modifica tipo macchina.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: il tipo macchina creato viene inserito nel database. I dati del tipo macchina modificato vengono aggiornati nel database.
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi al tipo macchina che si vuole creare/modificare.
  2. L’applicazione controlla che i dati inseriti dall’utente siano coerenti.
  3. Vengono inseriti/modificati i dati del nuovo tipo macchina/tipo macchina da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di creazione/modifica non sia andata a buon fine.

### **3.4.12 UC12 – Eliminazione Tipo Macchina**

* **Descrizione**: eliminazione tipo macchina.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. Il tipo macchina che si vuole cancellare deve rispettare i seguenti requisiti:
  1. Non deve essere assegnato a nessun prodotto.
  2. Non deve essere assegnato a nessuna macchina.
* **Post** **condizioni**: il tipo macchina viene eliminato dal database.
* **Procedimento**:
  1. L’utente seleziona il tipo macchina che vuole eliminare.
  2. L’applicazione controlla che il tipo macchina da cancellare rispetti i requisiti sopra esposti.
  3. Viene eliminato il tipo macchina dal db.
  4. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora il tipo macchina non possa essere eliminato per i motivi sopra citati.

### **3.4.13 UC13 – Visualizzazione Macchine**

* **Descrizione**: visualizzazione in tabella delle macchine.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: i dati delle macchine vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione dei dati delle macchine.
  2. Vengono letti dal database i dati relativi alle macchine
  3. Vengono mostrati a schermo i dati relativi alle macchine tramite una tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina di visualizzazione delle macchine non sia disponibile.

### **3.4.14 UC14 – Creazione/Modifica Macchina**

* **Descrizione**: creazione/modifica macchina.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: la macchina creata viene inserita nel database. I dati della macchina modificata vengono aggiornati nel database.
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi alla macchina che si vuole creare/modificare.
  2. L’applicazione controlla che i dati inseriti dall’utente siano coerenti.
  3. Vengono inseriti/modificati i dati della nuova macchina/ macchina da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di creazione/modifica non sia andata a buon fine.

### **3.4.15 UC15 – Eliminazione Macchina**

* **Descrizione**: eliminazione macchina.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: la macchina viene eliminata dal database.
* **Procedimento**:
  1. L’utente seleziona la macchina che vuole eliminare.
  2. Viene eliminata la macchina dal db.
  3. Viene mostrato a schermo l’esito dell’operazione.

### **3.4.16 UC16 – Visualizzazione Ordini Produzione non chiusi**

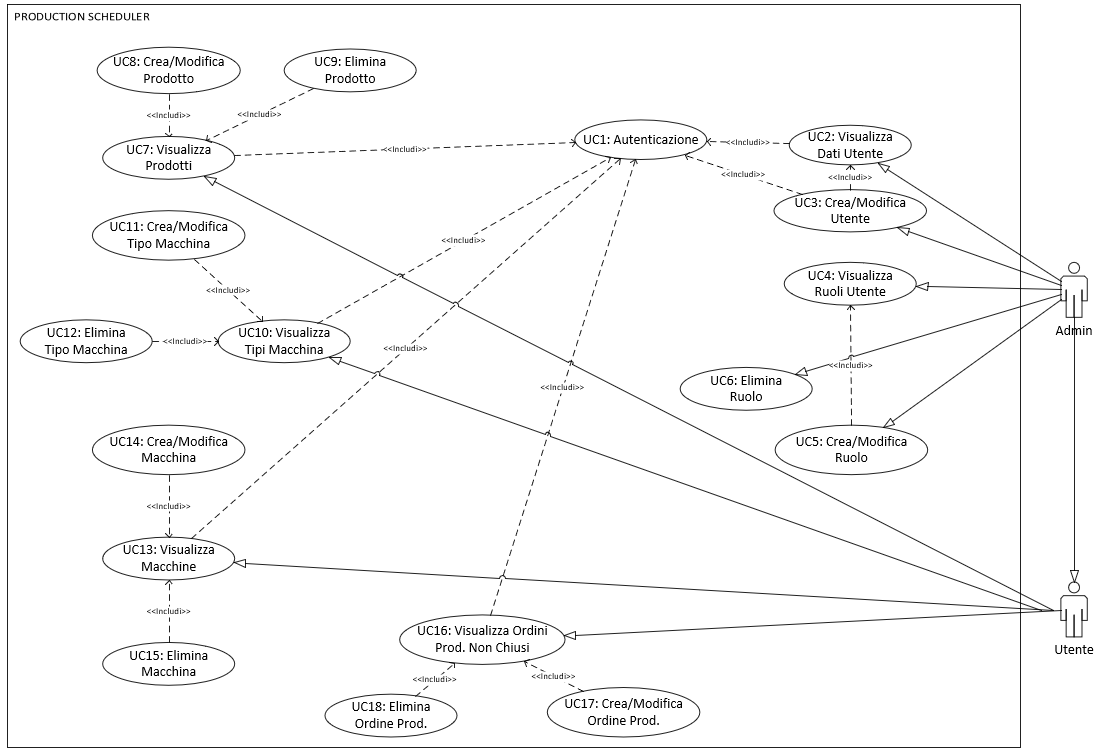
* **Descrizione**: visualizzazione in tabella degli ordini di produzione non chiusi.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: i dati degli ordini di produzione non chiusi vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione degli ordini di produzione non chiusi.
  2. Vengono letti dal database i dati relativi agli ordini di produzione non chiusi.
  3. Vengono mostrati a schermo i dati relativi agli ordini di produzione non chiusi tramite una tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina di visualizzazione degli ordini di produzione non chiusi non sia disponibile.

### **3.4.17 UC17 – Creazione/Modifica Ordine Produzione**

* **Descrizione**: creazione/modifica ordine produzione.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. In caso di modifica devono essere rispettate le seguenti condizioni:
  1. Il numero ordine non può essere modificato.
  2. Non si possono modificare ordini di produzione chiusi.
* **Post** **condizioni**: l’ordine di produzione viene inserito nel database con stato “Aperto”. I dati dell’ordine di produzione modificato vengono aggiornati nel database.
* **Procedimento**:
  1. L’utente inserisce/modifica i dati relativi all’ordine di produzione che si vuole creare/modificare.
  2. L’applicazione controlla che i dati inseriti dall’utente siano coerenti e che siano rispettate le condizioni esposte precedentemente.
  3. Vengono inseriti/modificati i dati del nuovo ordine di produzione/ordine di produzione da modificare nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
  5. Vengono mostrati a schermo i dati aggiornati letti dal db.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di creazione/modifica non sia andata a buon fine.

### **3.4.18 UC18 – Eliminazione Ordine di Produzione**

* **Descrizione**: eliminazione ordine di produzione.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. L’ordine di produzione che si vuole cancellare deve rispettare i seguenti requisiti:
  1. Non deve essere in stato chiuso.
* **Post** **condizioni**: l’ordine di produzione viene eliminato dal database.
* **Procedimento**:
  1. L’utente seleziona l’ordine di produzione che vuole eliminare.
  2. L’applicazione controlla che l’ordine di produzione da cancellare rispetti i requisiti sopra esposti.
  3. Viene eliminato l’ordine di produzione dal db.
  4. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’ordine di produzione non possa essere eliminato per i motivi sopra citati.

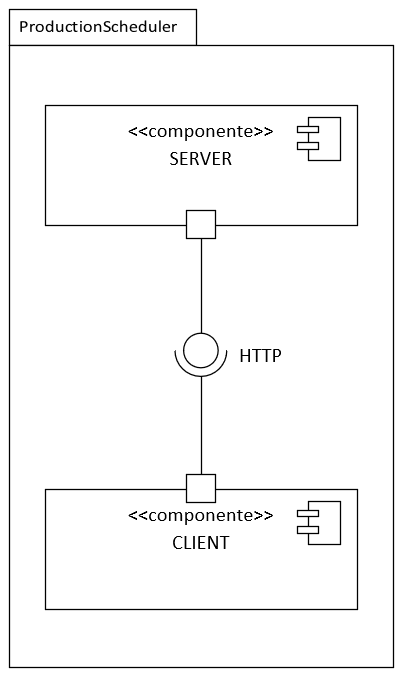


## **3.5 Modello Architetturale**

### **3.5.1 Component Diagram**

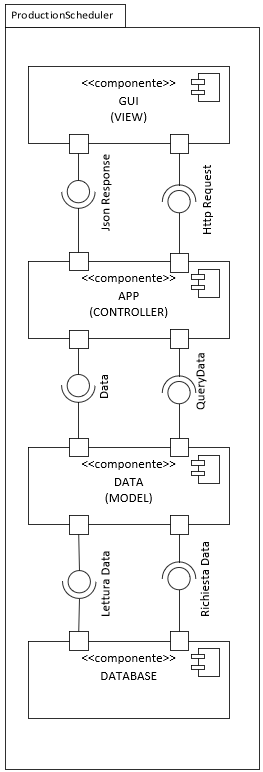
Ad alto livello il sistema è rappresentabile da due soli componenti: il Client che richiede un servizio e il Server che lo fornisce. I due componenti interagiscono tramite chiamate http.

Di seguito il Component Diagram del sistema ad Alto Livello.

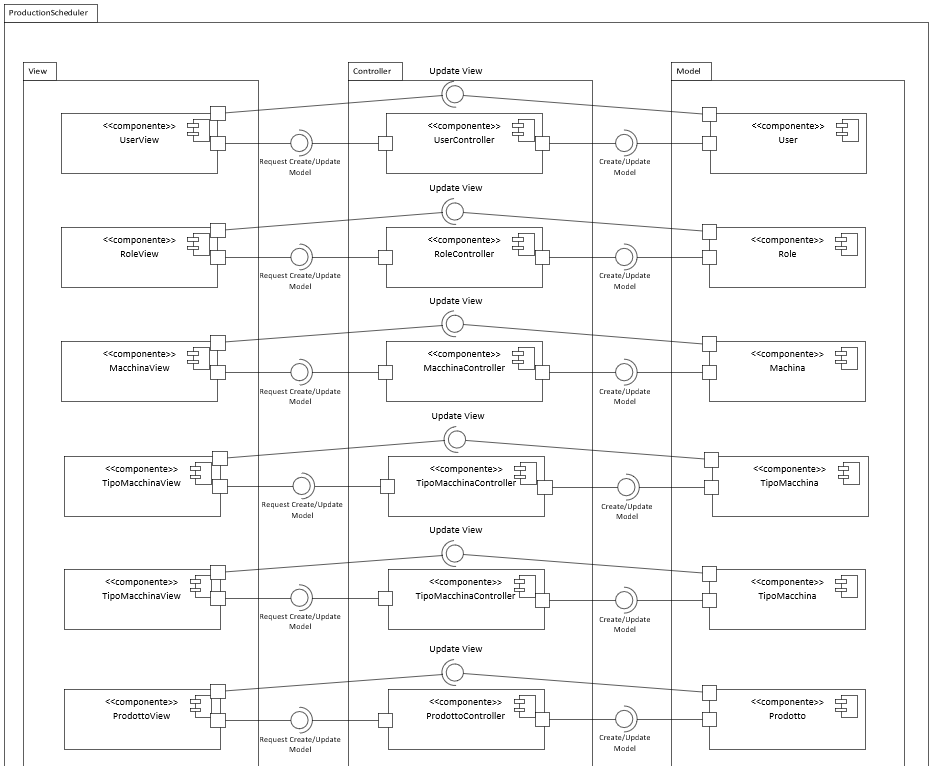


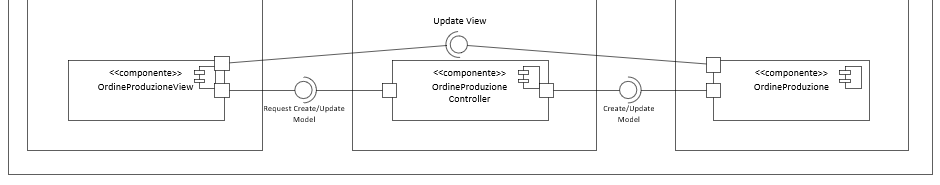
Ad un livello più basso si riescono a distinguere ulteriori componenti: GUI, APP, DATA, DATABASE. Grazie a questa ulteriore distinzione si riesce ad intuire meglio il flusso delle operazioni svolte dall’applicazione.

Il componente APP riceve le richieste provenienti dal componente GUI (interfaccia con la quale l’utente interagisce con l’applicazione). Il componente DATA invece ha il compito di interfacciarsi con il componente DATABASE per interagire direttamente con i dati dell’applicazione salvati nel database. In questo livello si possono inoltre individuare i componenti principali del design pattern MVC che verrà utilizzato per l’applicazione (Model, View, Controller).



Se scendiamo ancora più nel dettaglio possiamo notare i componenti principali (Model, View e Controller) contengono a loro volta componenti più specifici che svolgono un ruolo ben distinto all’interno dell’applicazione.





## **3.6 Programma di test**

Verranno principalmente testate le porzioni di codice non fornite da terze parti, assumendo che le funzionalità messe a disposizione siano già state testate dagli sviluppatori.

Il programma di test sarà composto di:

* Analisi statica del codice tramite il tool PSalm.
* Casi di test per ogni classe di tipo Model e Controller tramite PHPUnit.
* Verifica di Code Coverage tramite PHPUnit.
* Simulazione di utilizzo dell’applicazione da parte di un utente.
* Utilizzo di Postman per le chiamate http REST.

### **3.6.1 Utilizzo di Postman**

In fase di programmazione delle funzionalità dell’applicazione ci siamo affidati all’utilizzo di Postman per poter testare le API in modo più rapido e intuitivo.

L’utilizzo di Postman ci è risultato particolarmente utile in quanto permette di eseguire chiamate REST senza dover utilizzare il front-end dell’applicazione.

# **4. Iterazione 1**

Nell’iterazione 1 sono stati implementati i requisiti funzionali analizzati nell’iterazione 0.

Come da analisi, l’applicazione è stata sviluppata attraverso il framework Laravel che fornisce una struttura di progetto e una numerosa serie di librerie che aiutano lo sviluppatore nella costruzione di un software di tipo MVC.

La struttura MVC come detto in precedenza prevede tre elementi fondamentali:

1. **Model**, contiene i metodi di accesso ai dati, è il componente che si interfaccia direttamente con il database per recuperare, creare e modificare i dati.
2. **View**, è il componente che si occupa di visualizzare i dati all’utente e gestisce l’interazione fra quest’ultimo e l’infrastruttura sottostante.
3. **Controller,** è il componente posto a metà tra il Model e la View. Contiene metodi pubblici chiamati **action (API)** contenenti operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato nella View.

Le classi implementate per rappresentare i Model e i Controller necessari al funzionamento dell’applicazione sono state scritte utilizzando il linguaggio di programmazione PHP.

Le View invece sono state implementate utilizzando il motore di template PHP di Laravel, **Blade.**

Le View Blade sono compilate in linguaggio PHP che consente di scrivere codice PHP direttamente nella View.

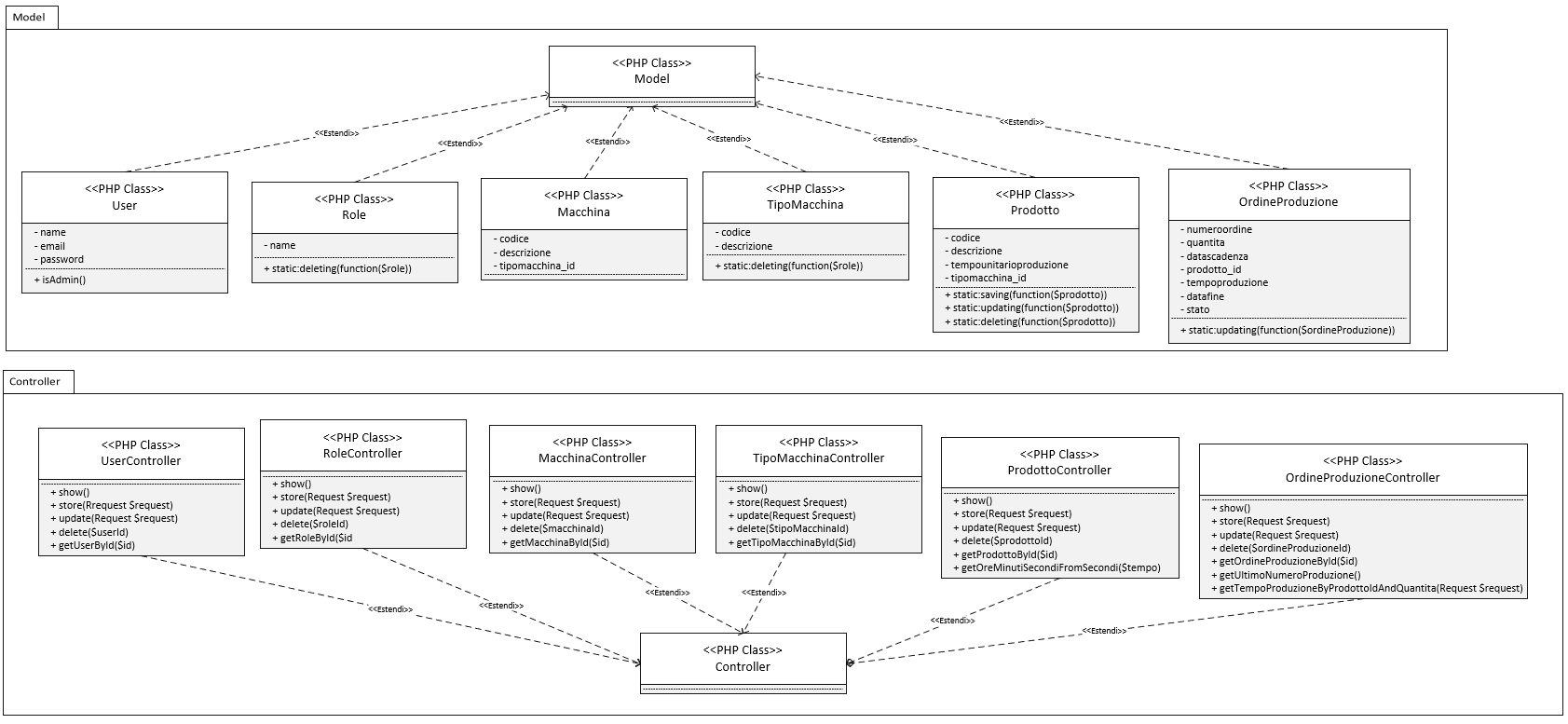
I linguaggi utilizzati per implementare le View sono quindi:

* PHP
* HMTL
* CSS
* JavaScript

Seguendo la struttura del design pattern architetturale MVC (Model View Controller) sono stati implementate le seguenti classi tramite linguaggio PHP:

* **Model:**
  1. User
  2. Role
  3. Prodotto
  4. Tipo Macchina
  5. Macchina
  6. OrdineProduzione
* **Controller:**
  1. UserController
  2. RoleController
  3. ProdottoController
  4. TipoMacchinaController
  5. MacchinaController
  6. OrdineProduzioneController

## **4.1 UML Class Diagram**

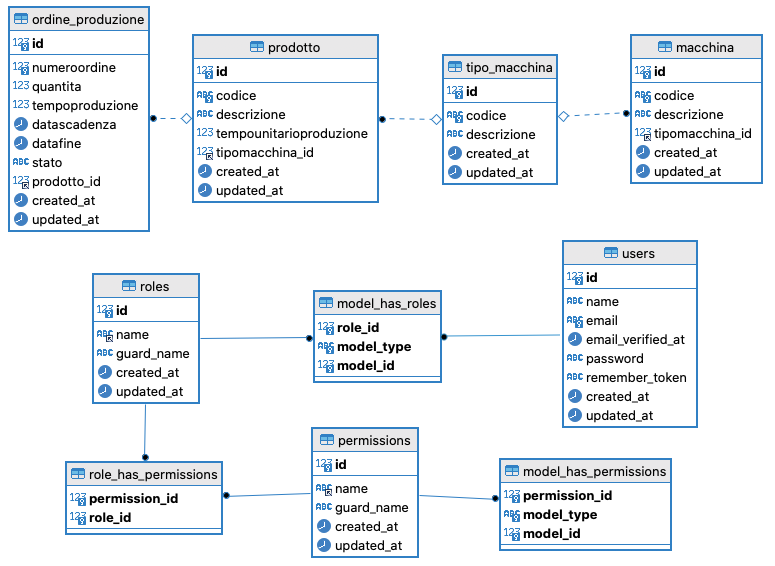


## **4.2 Progettazione Database**

L’applicazione per funzionare ha bisogno di un database nel quale poter salvare tutti i dati necessari. È fondamentale quindi progettare in maniera corretta e precisa la struttura delle tabelle che vanno a comporre il Database e delle relazioni che intercorrono tra esse.

Per la gestione dei dati dell’applicazione viene utilizzato MySQL, un RDBMS (Relational Database Management System) Open Source.

### **4.2.1 Modello E/R Database**

****

## **4.3 Testing**

### **4.3.1 Analisi statica**

Durante lo sviluppo dell’applicazione è stato utilizzato il tool **Psalm** per l’analisi statica del codice in modo da monitorare e prevenire gli errori di runtime.

Psalm viene aggiunto all’interno dell’applicazione tramite Composer e consente di essere personalizzato tramite il file di configurazione “psalm.xml”. All’interno del file di configurazione può essere specificato il livello di controllo sull’errore (8 il meno restrittivo, 1 il più restrittivo) e l’elenco di directory sulle quali il tool eseguirà l’analisi.

Immagine che contiene testo

Descrizione generata automaticamente

Di seguito un esempio di analisi eseguita alla fine dello sviluppo dell’iterazione 1.

Immagine che contiene testo

Descrizione generata automaticamente

### **4.3.2 Analisi Dinamica**

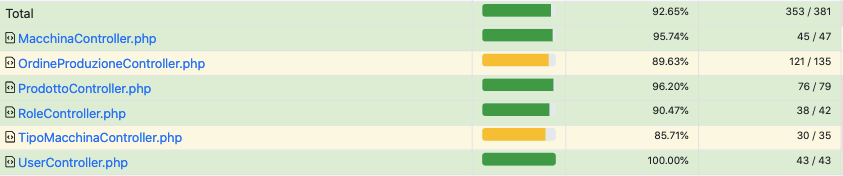
In questa prima iterazione sono state sottoposte a unit test le classi scritte manualmente da noi. I test sulle classi fornite da terze parti non sono stati effettuati, in quanto già testate dai loro sviluppatori prima di essere rese disponibili.

Di seguito vengono riportate le classi con i metodi testati:

* UserController
  1. public function \_\_construct()
  2. public function show()
  3. public function store(Request $request)
  4. public function update(Request $request)
  5. public function getUserById($id)
* RoleController
  1. public function \_\_construct()
  2. public function show()
  3. public function store(Request $request)
  4. public function update(Request $request)
  5. public function delete($roleId)
  6. public function getRoleById($id)
* MacchinaController
  1. public function show()
  2. public function store(Request $request)
  3. public function update(Request $request)
  4. public function delete($macchinaId)
  5. public function getMacchinaById($id)
* TipoMacchinaController
  1. public function show()
  2. public function store(Request $request)
  3. public function update(Request $request)
  4. public function delete($tipoMacchinaId)
  5. public function getTipoMacchinaById($id)
* ProdottoController
  1. public function show()
  2. public function store(Request $request)
  3. public function update(Request $request)
  4. public function delete($prodottoId)
  5. public function getProdottoById($id)
  6. protected function getOreMinutiSecondiFromSecondi($tempo)
* OrdineProduzioneController
  1. public function show()
  2. public function store(Request $request)
  3. public function update(Request $request)
  4. public function delete($ordineProduzioneId)
  5. public function getOrdineProduzioneById($id)
  6. public function getUltimoNumeroProduzione()
  7. protected function getTempoProduzioneByProdottoIdAndQuantita(Request $request)
  8. protected function getOreMinutiSecondiFromSecondi($tempo)
* User
  1. public function isAdmin()
* Role
  1. public static function boot()
* TipoMacchina
  1. public static function boot()
* Prodotto
  1. public static function boot()
* OrdineProduzione
  1. public static function boot()

Per eseguire i test è stato utilizzato il tool PHPUnit. Per la visualizzazione dei risultati di code coverage invece ci siamo affidati al tool PCOV. Questo tool lavorando inisieme a PHPUnit produce un file html contenente i risultati dei test di unità e consultabili attraverso il browser.

Di seguito i risultati ottenuti dopo l’esecuzione dei test.





# **5. Iterazione 2**

Nell’iterazione 2 viene sviluppata la procedura che consente la creazione della pianificazione di produzione sulla base delle macchine disponibili e degli ordini di produzione che si devono produrre.

Vengono inserite alcune funzionalità per migliorare la gestione degli ordini di produzione e una pagina per la gestione dei dati personali degli utenti che ora hanno la possibilità di modificare le proprie credenziali di autenticazione.

## **5.1 Specifiche funzionali**

Le specifiche funzionali introdotte in questa seconda iterazione sono:

|  |
| --- |
| FU8 – Gestione Profilo Utente |
| * L’utente loggato ha la possibilità di gestire i propri dati personali (nome, mail, password): * Visualizzazione dati personali * Modifica nome, email e password |

|  |
| --- |
| FU9 – Chiusura Ordini Produzione |
| * Un ordine di produzione può essere chiuso e “archiviato”, in modo da non essere preso in considerazione durante la pianificazione della produzione. |

|  |
| --- |
| FU10 – Riapertura Ordini Produzione |
| * Un ordine di produzione può essere riaperto, in modo da essere preso di nuovo in considerazione durante la pianificazione della produzione. |

|  |
| --- |
| FU11 – Visualizzazione Ordini Produzione Chiusi |
| * Gli utenti possono visualizzare lo storico di tutti gli Ordini di Produzione che sono stati prodotti e di conseguenza chiusi. |

|  |
| --- |
| FU12 – Gestione Pianificazione Produzione |
| * Gli utenti possono eseguire le seguenti operazioni di gestione della pianificazione di produzione: * Creare una nuova pianificazione di produzione sulla base degli ordini di produzione attualmente aperti e delle macchine disponibili. * Visualizzare le pianificazioni disponibili a DB. * Eliminare le pianificazioni di produzione. |

|  |
| --- |
| FU13 – Creazione Pianificazione Produzione |
| * Viene eseguita la procedura che genera una nuova pianificazione di produzione a partire dagli ordini di produzione aperti e dalle macchine disponibili. Vengono salvati a db tutti i dati coinvolti nella generazione della pianificazione in modo che persistano nel tempo e possano essere visualizzati dagli utenti. |

## **5.2 Casi d’uso**

### **5.2.1 UC19 – Visualizza dati Profilo**

* **Descrizione**: visualizza dati profilo utente loggato.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: vengono visualizzati in una pagina i dati relativi all’utente.
* **Procedimento**:
  1. L’utente richiede la visualizzazione dei propri dati personali.
  2. Vengono letti dal database i dati relativi all’utente.
  3. Vengono mostrati a schermo i dati dell’utent.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina non sia disponibile o l’utente non dispone delle autorizzazioni necessarie.

### **5.2.2 UC20 – Modifica dati Profilo**

* **Descrizione**: modifica dati utente loggato.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: viene mostrato a schermo il risultato dell’operazione. I dati dell’utente modificati vengono aggiornati nel database.
* **Procedimento**:
  1. L’utente modifica i propri dati personali.
  2. L’applicazione controlla che i dati inseriti dall’utente siano coerenti e che siano rispettate le seguenti condizioni:
     + Non possono esistere due utenti con la stessa mail.
  3. Vengono aggiornati i dati dell’utente nel db.
  4. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di modifica non sia andata a buon fine.

### **5.2.3 UC21 – Chiusura ordini di produzione aperti**

* **Descrizione**: chiusura di un ordine di produzione aperto.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. L’ordine di produzione che si vuole chiudere deve essere aperto.
* **Post** **condizioni**: lo stato dell’ordine di produzione viene aggiornato nel db e viene impostato a “C”.
* **Procedimento**:
  1. L’utente seleziona l’ordine di produzione che vuole chiudere.
  2. Viene impostato lo stato dell’ordine di produzione a “C” nel db.
  3. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di chiusura dell’ordine non sia andata a buon fine.

### **5.2.4 UC22 – Visualizzazione Ordini Produzione chiusi**

* **Descrizione**: visualizzazione in tabella degli ordini di produzione chiusi.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: i dati degli ordini di produzione chiusi vengono visualizzati in una tabella.
* **Procedimento**:
  1. L’utente richiede la visualizzazione degli ordini di produzione chiusi.
  2. Vengono letti dal database i dati relativi agli ordini di produzione chiusi.
  3. Vengono mostrati a schermo i dati relativi agli ordini di produzione chiusi tramite una tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora la pagina di visualizzazione degli ordini di produzione chiusi non sia disponibile.

### **5.2.5 UC23 – Riapertura Ordine Produzione Chiuso**

* **Descrizione**: riapertura di un ordine di produzione chiuso.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato. L’ordine di produzione che si vuole chiudere deve essere chiuso.
* **Post** **condizioni**: lo stato dell’ordine di produzione viene aggiornato nel db e viene impostato a “A”.
* **Procedimento**:
  1. L’utente seleziona l’ordine di produzione che vuole riaprire.
  2. Viene impostato lo stato dell’ordine di produzione a “A” nel db.
  3. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di riapertura dell’ordine non sia andata a buon fine.

### **5.2.6 UC24 – Creazione Pianificazione Produzione**

* **Descrizione**: creazione della pianificazione di produzione sulla base delle macchine disponibili e degli ordini di produzione aperti.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: viene creata una pianificazione di produzione assegnando gli ordini di produzione alle macchine disponibili. Vengono salvati su db i dati necessari alla rappresentazione della pianificazione di produzione.
* **Procedimento**:
  1. L’utente inserisce un nome simbolico per identificare la pianificazione di produzione, una descrizione facoltativa e la data di inizio stimata della produzione.
  2. L’applicazione esegue il programma incaricato alla creazione della pianificazione di produzione.
  3. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione di creazione della pianificazione di produzione non vada a buon fine.

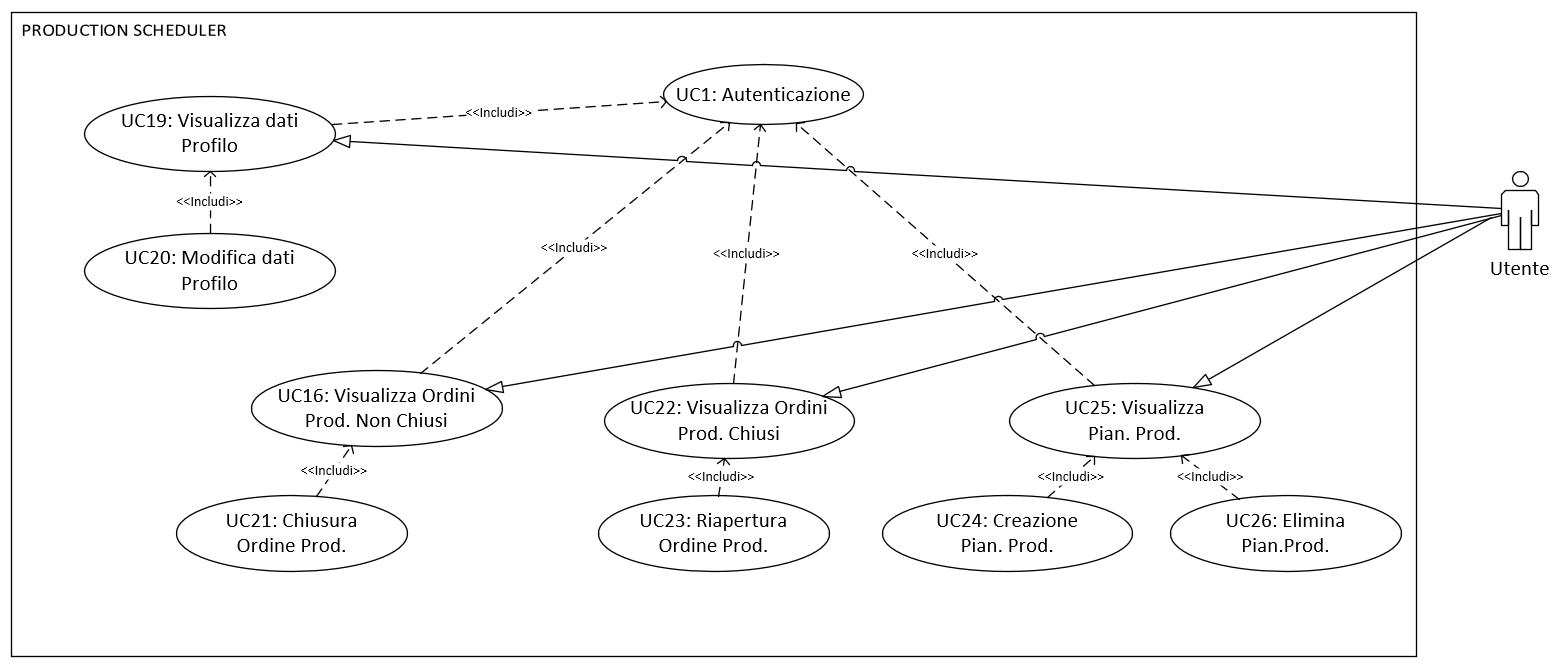
### **5.2.7 UC25 – Visualizzazione Pianificazione di Produzione**

* **Descrizione**: visualizzazione della Pianificazione di Produzione richiesta dall’utente.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: viene mostrata a scherma tramite una tabella la pianificazione di produzione richiesta.
* **Procedimento**:
  1. L’utente seleziona il nome della pianificazione di produzione che vuole visualizzare.
  2. Vengono letti i dati a db relativi alla pianificazione richiesta.
  3. Vengono mostrati a schermo i dati relativi alla pianificazione di produzione richiesta tramite tabella.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore la pagina richiesta non sia disponibile.

### **5.2.8 UC26 – Elimina Pianificazione di Produzione**

* **Descrizione**: eliminazione pianificazione produzione.
* **Attori** **coinvolti**: utente.
* **Precondizioni**: l’utente deve essere autenticato.
* **Post** **condizioni**: la pianificazione di produzione viene eliminata dal database.
* **Procedimento**:
  1. L’utente seleziona la pianificazione di produzione che vuole eliminare.
  2. L’applicazione esegue la procedura per eliminare dal db la pianificazione di produzione e tutti i dati ad essa collegati.
  3. Viene mostrato a schermo l’esito dell’operazione.
* **Eccezioni**:
  1. Viene visualizzato un messaggio di errore qualora l’operazione non sia andata a buon fine.

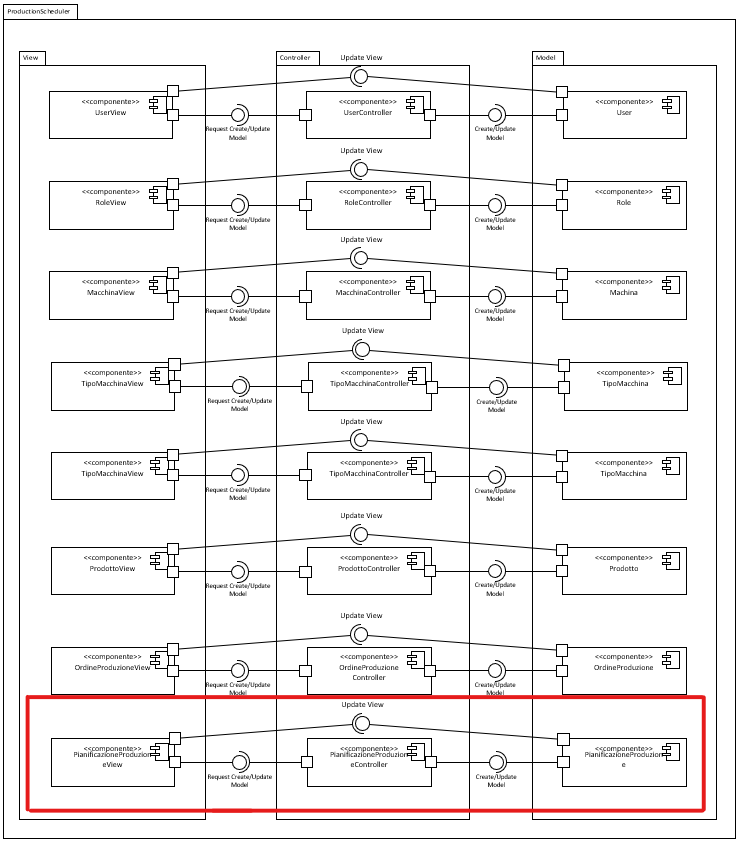
Di seguito la sintesi dei casi d’uso aggiunti in questa seconda iterazione.



## **5.3 Modello Architetturale**

### **5.3.1 Component Diagram**

Rispetto all’iterazione precedente sono stati aggiunti i componenti per gestire la pianificazione di produzione che sono visibili guardando nel dettaglio i componenti del sistema. Il component diagram aggiornato diventa quindi il seguente, dove vengono evidenziati i tre componenti aggiunti rispetto all’iterazione precedente.



Seguendo la struttura del design pattern architetturale MVC, in questa seconda iterazione sono stati implementate le seguenti classi tramite linguaggio PHP:

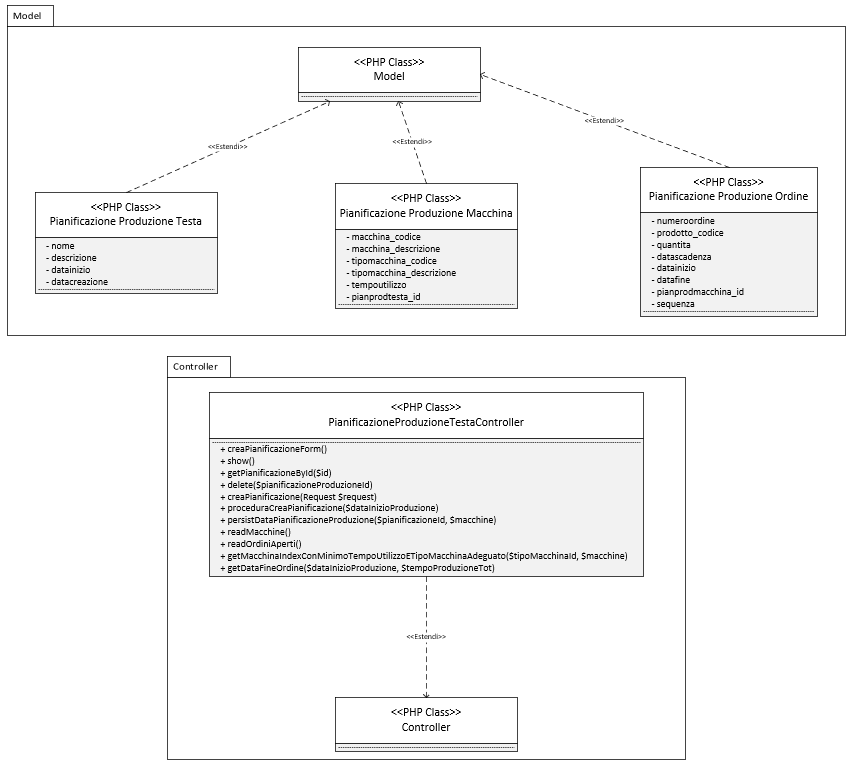
* **Model:**

1. PianificazioneProduzioneTesta
2. PianificazioneProduzioneMacchina
3. PianificazioneProduzioneOrdine

* **Controller:**
  1. PianificazioneProduzioneTestaController

### **5.3.2 UML Class Diagram**

Di seguito il diagramma delle classi aggiunte nell’iterazione 2.



## **5.4 Progettazione Database**

Per poter implementare le funzionalità previste nell’iterazione 2, la struttura del database è stata modificata con l’aggiunta di tre tabelle utilizzate per il salvataggio dei dati che vengono ottenuti dalla procedura di calcolo della pianificazione di produzione.

Le tre tabelle aggiunte sono le seguenti:

* pianificazione\_produzione\_testa
* pianificazione\_produzione\_macchina
* pianificazione\_produzione\_ordine

### **5.4.1 Modello E/R Database**

Le uniche modifiche al modello E/R rispetto all’iterazione precedente sono l’aggiunta delle seguenti 3 tabelle.



## **5.5 Testing**

### **5.5.1 Analisi Statica**

Come durante l’iterazione 1 per eseguire l’analisi statica del codice è stato utilizzato il tool **Psalm.**

Di seguito un esempio di analisi eseguita alla fine dello sviluppo dell’iterazione 1.

Immagine che contiene testo

Descrizione generata automaticamente

### **5.5.2 Analisi Dinamica**

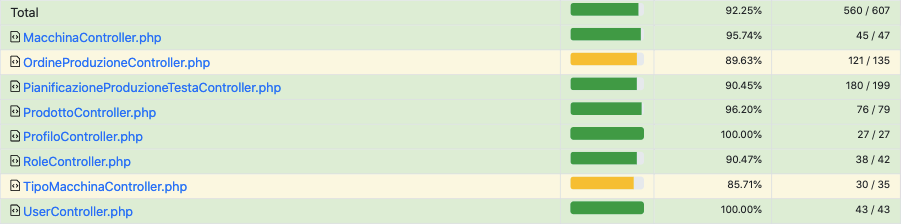
Anche nella seconda iterazione sono state sottoposte a unit test le classi scritte manualmente da noi. Anche in questo caso i test sulle classi fornite da terze parti non sono stati effettuati, in quanto già testate dai loro sviluppatori prima di essere rese disponibili.

Tutte le classi e i metodi già precedentemente testati sono stati testati ancora per controllare che le nuove modifiche non abbiano introdotto errori.

Di seguito vengono riportate le classi e i metodi testati che sono state aggiunte rispetto all’iterazione 1.

* ProfiloController:
  1. public function show()
  2. public function updateNameMail(Request $request)
  3. public function updatePassword(Request $request)
* OrdineProduzioneController:
  1. public function close( $ordineProduzioneId)
  2. public function open( $ordineProduzioneId)
  3. public function showOrdiniChiusi()
* PianificazioneProduzioneTestaController:
  1. public function creaPianificazioneForm()
  2. public function show()
  3. public function getPianificazioneById($id)
  4. public function delete($pianificazioneProduzioneId)
  5. public function creaPianificazione(Request $request)
  6. protected function proceduraCreaPianificazione($dataInizioProduzione)
  7. protected function persistDataPianificazioneProduzione($pianificazioneId, $macchine)
  8. protected function readMacchine()
  9. protected function readOrdiniAperti()
  10. protected getMacchinaIndexConMinimoTempoUtilizzoETipoMacchinaAdeguato($tipoMacchinaId, $macchine)
  11. protected function getDataFineOrdine($dataInizioProduzione, $tempoProduzioneTot)

Risultati test.





# **6. Iterazione 3**

Nell’iterazione 3 è stata svolta una revisione del codice per facilitarne la lettura e la comprensione.

Sono stati sistemati anche alcuni problemi nella User Interface.

Durante l’iterazione 3 è stata aggiunta anche l’implementazione di Docker per facilitare il deployment dell’applicazione.

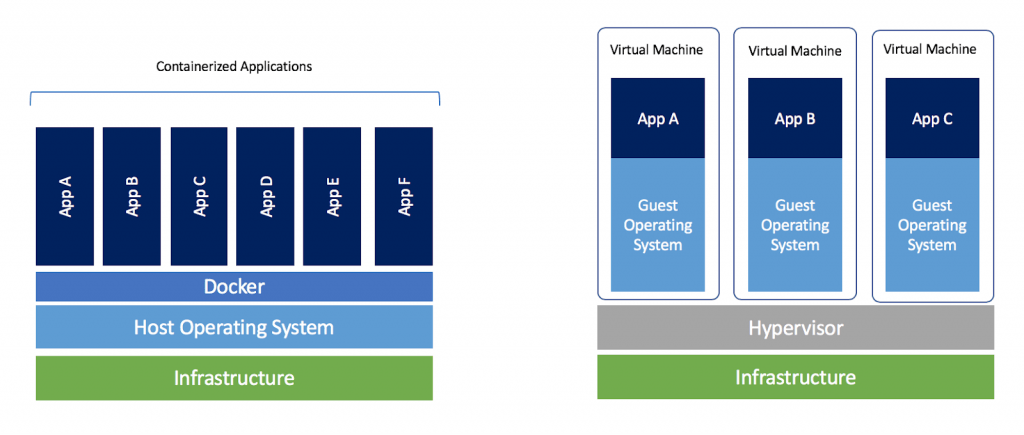
## **6.1 Docker**

Docker è un progetto open-source basato su contenitori Linux. Usa le funzionalità del kernel di Linux per creare container su un sistema operativo.

A differenza di una VM che fornisce la virtualizzazione dell’hardware, un container fornisce la virtualizzazione a livello di sistema operativo.

I contenitori hanno l’aspetto di una VM, infatti, dispongono di spazio privato per l’elaborazione, possono eseguire comandi come root, avere un’interfaccia di rete privata e un indirizzo IP, consentire percorsi personalizzati, ecc.

L’unica grande differenza tra container e macchine virtuali è che i contenitori condividono il kernel del sistema host con altri contenitori.



Dall’immagine sopra si può vedere che tutta l’architettura a livello di sistema operativo viene condivisa tra i container. Le uniche parti create da zero sono i container. Questo rende i container molto leggeri e veloci.

Vantaggi utilizzo Docker:

* **Facilità di utilizzo**: docker rende molto semplice sfruttare i container per creare e testare rapidamente applicazioni. Consente a chiunque di impacchettare un’applicazione in locale ed eseguirla in cloud o su un altro dispositivo senza doverla modificare per renderla compatibile.
* **Velocità**: i container di docker sono molto leggeri e veloci. È possibile creare ed eseguire un container Docker in pochi secondi rispetto al tempo richiesto normalmente per una macchina virtuale dato che quest’ultime devono avviare un sistema operativo virtuale completo.
* **Modularità e Scalabilità**: docker semplifica la suddivisione delle funzionalità dell’applicazione in singoli container. Ad esempio si può avere un db MySQL in esecuzione su un container, un webserver Nginx su un altro e l’applicazione vera e propria su un terzo container.

## **6.2 Implementazione Docker**

Per utilizzare l’applicazione tramite Docker è stato necessario costruire tre container:

* Un container sul quale eseguire un database MySQL
* Un container per un webserver Nginx in grado di gestire le richieste di trasferimento di pagine web di un client.
* Un container basato su PHP per l’esecuzione dell’applicazione vera e propria.

Le immagini dei container per il database e per il webserver sono state prese da DockerHub, un servizio fornito da docker per trovare e condividere le immagini dei container.

L’immagine utilizzata per il container dell’applicazione è definita nel **Dockerfile** presente nella repository principale del progetto.

Questo Dockerfile imposterà l’immagine di base del container e specificherà i comandi e le istruzioni necessari per creare l’immagine dell’applicazione Laravel.

Di seguito le istruzioni presenti nel Dockerfile.

FROM php:7.2-fpm  
  
COPY composer.lock composer.json /var/www/  
WORKDIR /var/www  
  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 libpng-dev \  
 libjpeg62-turbo-dev \  
 libfreetype6-dev \  
 locales \  
 zip \  
 jpegoptim optipng pngquant gifsicle \  
 vim \  
 unzip \  
 git \  
 curl  
  
RUN apt-get clean && rm -rf /var/lib/apt/lists/\*  
  
RUN docker-php-ext-install pdo\_mysql mbstring zip exif pcntl  
RUN docker-php-ext-configure gd --with-gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ --with-png-dir=/usr/include/  
RUN docker-php-ext-install gd  
  
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer  
  
RUN groupadd -g 1000 www  
RUN useradd -u 1000 -ms /bin/bash -g www www  
  
COPY . /var/www  
  
COPY --chown=www:www . /var/www  
  
USER www  
  
EXPOSE 9000  
CMD [**"php-fpm"**]

Nel file **docker-compose.yml** invece vengono organizzati e documentati tutti i container necessari all’applicazione.

Il tool Docker Compose si occupa invece di interpretare il file **docker-compose-yml** e instanziare i container in esso contenuti.

version: **'3'**services:  
  
 *#PHP Service* app:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 container\_name: productionscheduler\_app  
 restart: unless-stopped  
 tty: true  
 environment:  
 SERVICE\_NAME: app  
 SERVICE\_TAGS: dev  
 working\_dir: /var/www  
 volumes:  
 - ./:/var/www  
 networks:  
 - app-network  
  
 *#Nginx Service* webserver:  
 image: nginx:alpine  
 container\_name: productionscheduler\_webserver  
 restart: unless-stopped  
 tty: true  
 ports:  
 - **"8080:80"** - **"444:443"** volumes:  
 - ./:/var/www  
 - ./docker/nginx/conf.d/:/etc/nginx/conf.d/  
 networks:  
 - app-network  
  
 *#MySQL Service* db:  
 image: mysql:5.7.22  
 container\_name: productionscheduler\_db  
 restart: unless-stopped  
 tty: true  
 ports:  
 - **"3306:3306"** environment:  
 MYSQL\_DATABASE: production\_scheduler  
 MYSQL\_ROOT\_PASSWORD: admin  
 SERVICE\_TAGS: dev  
 SERVICE\_NAME: mysql  
 volumes:  
 - ./dbdata:/var/lib/mysql/  
 networks:  
 - app-network  
  
*#Docker Networks*networks:  
 app-network:  
 driver: bridge  
*#Volumes*volumes:  
 dbdata:  
 driver: local

# **7. Algoritmi**

## **7.1 Obiettivo Production Planning**

La procedura di pianificazione della produzione ha come obiettivo quello di minimizzare il tempo di completamento degli ordini di produzione, massimizzando quindi l’efficienza della produzione.

Si hanno a disposizione i seguenti dati di input:

* m macchine di produzione
* n ordini di produzione

Ogni macchina m appartiene ad una tipologia ben precisa.

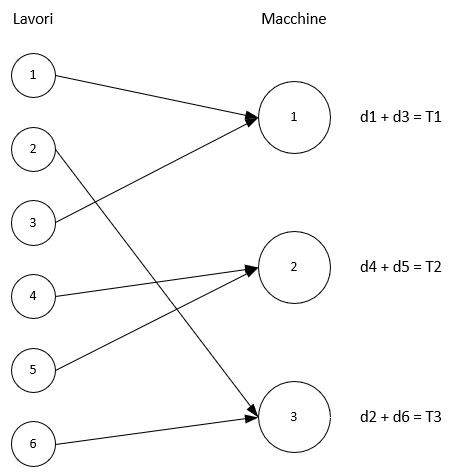
Ogni ordine di produzione ha i seguenti attributi:

* una durata di esecuzione.
* una data di scadenza.
* un prodotto che necessita di una macchina di una specifica tipologia per essere prodotto.

Il tempo di inizio della produzione è fornito dall’utente al momento della creazione della pianificazione.

L’obiettivo principale è dunque quello di suddividere gli ordini di produzione tra le varie macchine affinché vengano completati nel minor tempo possibile e cercando di rispettare la data di scadenza.

Un vincolo molto importante da tenere in considerazione riguarda la tipologia delle macchine alle quali assegnare un ordine di produzione, un ordine di produzione per un prodotto di una tipologia non compatibile con quella di una determinata macchina non potrà essere assegnato ad essa.



Tempo completamento di tutti i lavori:

L’obiettivo è dunque minimizzare t.

## **7.2 Utilizzo tecnica Greedy**

Per raggiungere l’obiettivo del Production Planning viene utilizzata una tecnica Greedy

Inizializzazione:

* tutte le macchine m sono scariche, ossia il loro tempo totale di utilizzo è 0
* nessun ordine di produzione è assegnato ad alcuna macchina
* gli ordini di produzione vengono ordinati in ordine crescente in funzione della data di scadenza (verranno assegnati alle macchine gli ordini che hanno una scadenza più prossima) e in ordine decrescente in funzione della durata di esecuzione (a parità di scadenza vengono assegnati prima gli ordini che cubano il tempo maggiore di esecuzione)

Iterazione:

* viene selezionato l’ordine di produzione *i* tra quelli non ancora assegnati
* viene selezionata la macchina *j* con il tempo totale di utilizzo più basso e tipologia compatibile con l’ordine di produzione *i*.
* viene assegnato l’ordine di produzione *i* alla macchina *j* e si incrementa il tempo totale di utilizzo di tale macchina per un valore pari alla durata di esecuzione dell’ordine di produzione.

Terminazione: tutti gli ordini di produzione sono assegnati alle macchine.

Problemi: si verifica un errore qualora non sia presente una macchina con tipologia compatibile all’ordine di produzione preso in esame. In questo caso l’ordine di produzione non può essere assegnato.

## **7.3 Algoritmo Pianificazione Produzione**

In questo paragrafo viene presentato lo pseudocodice dell’algoritmo di Pianificazione della Produzione.

Ogni macchina contiene i seguenti dati e viene rappresentata tramite un array associativo:

macchina = [

‘macchina\_id’,

‘tipomacchina\_id’,

‘tempoutilizzo’

‘ordiniProduzione’

]

Il ‘tempoutilizzo’ viene inizializzato a 0 per tutte le macchine.

Gli ‘ordiniProduzione’ sono a loro volta un array associativo inizializzato ad array vuoto che verrà popolato con gli ordini assegnati alla macchina.

Ogni ordine di produzione contiene i seguenti dati e viene rappresentato tramite un array associativo:

ordineProduzione = [

‘ordine\_id’,

‘numeroordine’,

‘tempoproduzione’,

‘datascadenza’,

‘tipomacchina\_id’

]

***Algoritmo 1 –*** *CreaPianificazioneProduzione()*

array macchine = [1 … n]

array ordiniProduzione = [1 … m]

*// vengono ordinati gli ordini di Produzione in senso crescente per data di scadenza e in senso decrescente per tempo di esecuzione*

ordiniProduzione = ordinamentoPerDataScadenzaETempoEsecuzione(array ordiniProduzione)

**foreach** ordiniProduzione **as** ordineProduzione **do**

*// viene recuperato l’indice della macchina a cui assegnare l’ordine con il minor tempo di utilizzo totale e che sia compatibile con l’ordine di produzione*

macchinaIndex = false

tempoMinimo = +infinito

**foreach** macchine **as** macchina **do**

**if** (macchina[‘tipomacchina\_id’] == tipoMacchina) **then**

**if** (

tempoMinimo == +infinito **or**

tempoMinimo > macchina[‘tempoutilizzo’]

) **then**

tempoMinimo = macchina[‘tempoutilizzo’]

macchinaIndex = indice macchina

**endif**

**endif**

**endforeach**

*// viene fatto un controllo se esiste una macchina compatibile*

**if** (macchinaIndex == false) **then**

non esistono macchine di tipo compatibile con l’ordine di produzione

**continue**

**endif**

***//*** *se esiste una macchina compatibile le assegno l’ordine di produzione e aumento il suo tempo di utilizzo di un valore pari al tempo di esecuzione dell’ordine*

macchine[macchinaIndex][‘ordiniProduzione’] = ordineProduzione

macchine[macchinaIndex][‘tempoutilizzo’] += ordineProduzione[‘tempoproduzione’]

**endforeach**

### **7.3.1 Analisi Complessità**

La prima operazione svolta dall’algoritmo è l’ordinamento degli ordini di Produzione che può essere svolto tramite un algoritmo di MergeSort che ha complessità dove *m* è il numero di ordini di produzione da ordinare.

L’algoritmo successivamente entra in un ciclo for che contiene al suo interno un altro ciclo for. Il ciclo for più esterno scandisce gli *m* ordini di produzione mentre quello interno scandisce le *n* macchine.

Il costo computazionale di questi due cicli for annidati è dunque dove *m* è il numero degli ordini di produzione e *n* è il numero di macchine.

Il costo complessivo dell’algoritmo è dunque maggiormente influenzato dai due cicli annidati, ne consegue che .

### **7.3.2 Implementazione Algoritmo Con Linguaggio PHP**

La procedura di creazione della pianificazione di produzione è stata implementata tramite linguaggio PHP all’interno della classe ‘PianificazioneProduzioneTestaController’.

public function creaPianificazione(Request $request) {  
 // validazione sui dati inseriti dall'utente  
 $request->validate([  
 **'nome'** => **'required|unique:pianificazione\_produzione\_testa,nome'**,  
 **'datainizio'** => **'required'** ]);  
  
 DB::beginTransaction();  
 try {  
 $nome = $request[**'nome'**];  
 $descrizione = empty($request[**'descrizione'**]) ? **''** : $request[**'descrizione'**];  
 $dataInizio = $request[**'datainizio'**];  
  
 //salvataggio Record Testa Pianificazione Produzione (tabella: pianificazione\_produzione\_testa)  
 $pianificazioneProduzioneTesta = new PianificazioneProduzioneTesta();  
 $pianificazioneProduzioneTesta->nome = $nome;  
 $pianificazioneProduzioneTesta->descrizione = $descrizione;  
 $pianificazioneProduzioneTesta->datacreazione = new \DateTime();  
 $pianificazioneProduzioneTesta->datainizio = $dataInizio;  
 $pianificazioneProduzioneTesta->save();  
  
 $pianificazioneId = $pianificazioneProduzioneTesta->id;  
  
 //procedura calcolo pianificazione produzione, ottengo la struttura dati da persistere a db  
 $macchine = $this->proceduraCreaPianificazione($dataInizio);  
  
 //salvataggio struttura dati su db nelle tabelle pianificazione\_produzione\_macchina e pianificazione\_produzione\_ordine  
 $this->persistDataPianificazioneProduzione($pianificazioneId, $macchine);  
   
 $request->session()->flash(**'status'**, **'Pianificazione Produzione creata correttamente'**);  
 DB::commit();  
 } catch (\Exception $e) {  
 DB::rollBack();  
 return new JsonResponse([**'errors'** => $e->getMessage()]);  
 }  
  
 return new JsonResponse([**'success'** => **'1'**]);  
}

Tutta la procedura è posta sotto transazione in modo tale che in caso di eventuali errori il database venga ripristinato allo stato precedente l’esecuzione della procedura e non rimangano dati ‘sporchi’ nelle tabelle.

Nella prima parte vengono acquisiti i dati inseriti dall’utente al momento della chiamata della procedura (nome, descrizione, data inizio produzione). Questi dati vengono utilizzati per creare a db il record di ‘intestazione’ della pianificazione di produzione.

Successivamente viene chiamata la funzione ‘*proceduraCreaPianificazione*’ che esegue effettivamente l’algoritmo per calcolare la pianificazione di produzione. La funzione ritorna la struttura contenente i dati della pianificazione che dovranno poi essere salvati nel database.

protected function proceduraCreaPianificazione($dataInizioProduzione) {  
 //questo array verrà modificato per tenere traccia della proceduara di calcolo della pianificazione  
 $macchine = $this->readMacchine();  
 //array contenente gli ordini aperti da schedulare sulle macchine  
 $ordiniAperti = $this->readOrdiniAperti();  
  
 //scandisco tutti gli ordini di produzione aperti per allocarli alle macchine disponibili  
 foreach ($ordiniAperti as $ordineAperto) {  
 //ottengo l'indice della macchina più scarica compatibile con il prodotto che si vuole produrre  
 $macchinaIndex = $this->getMacchinaIndexCompatibile($ordineAperto[**'tipomacchina\_id'**], $macchine);  
 //se non esiste una macchina alla quale assegnare un ordine di produzione tengo traccia dell'errore  
 if ($macchinaIndex === false) {  
 $this->erroreProcedura = true;  
 $this->elencoErroriProcedura[] = [  
 **'numeroordine'** => $ordineAperto[**'ordineproduzione\_numerordine'**],  
 **'tipomacchina\_codice'** => $ordineAperto[**'tipomacchina\_codice'**],  
 **'tipomacchina\_descrizione'** => $ordineAperto[**'tipomacchina\_descrizione'**],  
 ];  
 continue;  
 }  
  
 //assegnare ordine produzione alla macchina e aggiornare tempo produzione  
 $dataFine = $this->getDataFineOrdine(  
 $dataInizioProduzione,  
 $macchine[$macchinaIndex][**'tempoutilizzo'**] + $ordineAperto[**'ordineproduzione\_tempoproduzione'**]  
 );  
 $macchine[$macchinaIndex][**'tempoutilizzo'**] = $macchine[$macchinaIndex][**'tempoutilizzo'**] + $ordineAperto[**'ordineproduzione\_tempoproduzione'**];  
 $macchine[$macchinaIndex][**'ordiniproduzione'**][] = [  
 **'numeroordine'** => $ordineAperto[**'ordineproduzione\_numerordine'**],  
 **'prodotto\_codice'** => $ordineAperto[**'prodotto\_codice'**],  
 **'prodotto\_descrizione'** => $ordineAperto[**'prodotto\_descrizione'**],  
 **'quantita'** => $ordineAperto[**'ordineproduzione\_quantita'**],  
 **'datascadenza'** => $ordineAperto[**'ordineproduzione\_datascadenza'**],  
 **'datainizio'** => $dataInizioProduzione,  
 **'datafine'** => $dataFine  
 ];  
 }  
  
 //compongo la stringa degli errori da visualizzare lato client all'utente  
 if ($this->erroreProcedura) {  
 $errorMessage = **'Errore schedulazione Ordini di Produzione, non si dispone di una macchina a cui assegnare i seguenti ordini:'**;  
 foreach ($this->elencoErroriProcedura as $errore) {  
 $errorMessage = $errorMessage . **' '** . $errore[**'numeroordine'**] . **','**;  
 }  
 $errorMessage = substr($errorMessage, 0, -1);  
 throw new \Exception($errorMessage);  
 }  
  
 return $macchine;  
}

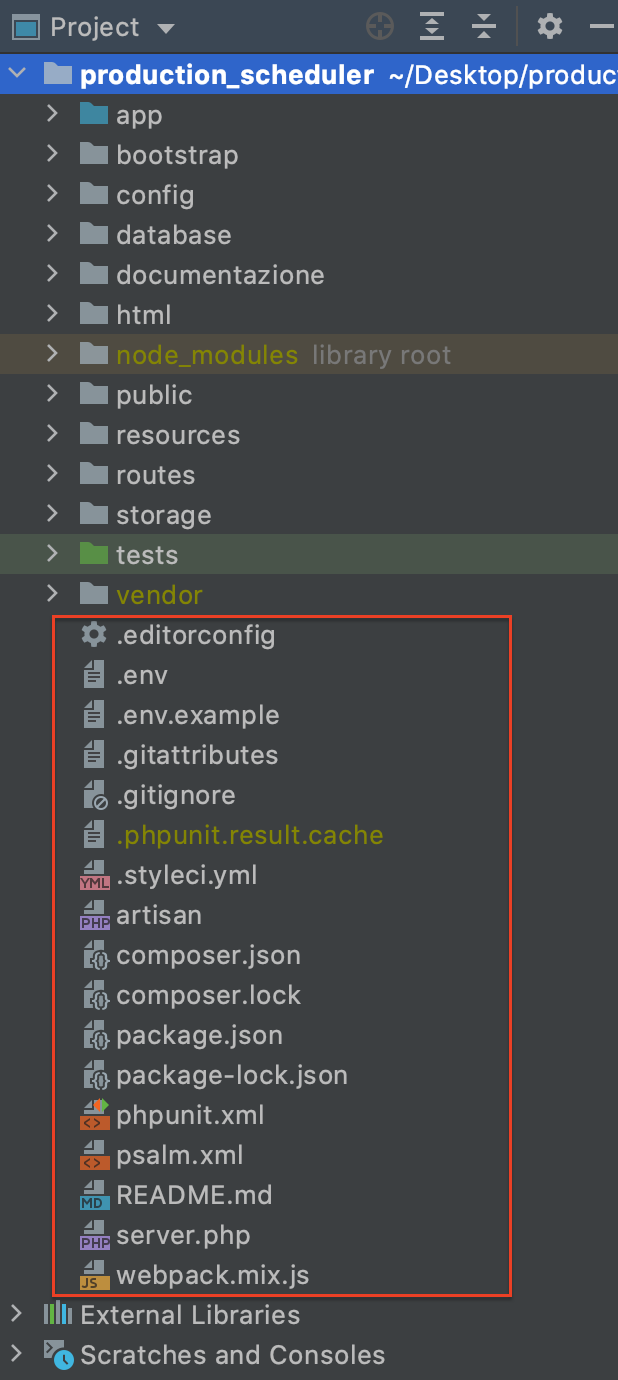
protected function getMacchinaIndexCompatibile($tipoMacchinaId, array $macchine) {  
 $tempoMinimo = false;  
 $macchinaIndex = false;  
 foreach ($macchine as $index => $macchina) {  
 if ($macchina[**'tipomacchina\_id'**] == $tipoMacchinaId) {  
 if ($tempoMinimo === false || ($tempoMinimo > $macchina[**'tempoutilizzo'**])) {  
 $tempoMinimo = $macchina[**'tempoutilizzo'**];  
 $macchinaIndex = $index;  
 }  
 }  
 }  
  
 return $macchinaIndex;  
}

Una volta che completata la funzione che esegue l’algoritmo per creare la pianificazione di produzione viene lanciata la funzione ‘*persistDataPianificazioneProduzione*’ che si occupa di salvare i dati sul database.

protected function persistDataPianificazioneProduzione($pianificazioneId, array $macchine): void {  
 //scandisco tutte le macchine a cui ho assegnato un ordine di produzione da produrre  
 foreach ($macchine as $macchina) {  
 //controllo che alla macchina siano assegnati degli ordini di produzione  
 if (array\_key\_exists(**'ordiniproduzione'**, $macchina)) {  
 //creo record tabella pianificazione\_produzione\_macchina  
 $pianificazioneProduzioneMacchina = new PianificazioneProduzioneMacchina();  
 $pianificazioneProduzioneMacchina->macchina\_codice = $macchina[**'macchina\_codice'**];  
 $pianificazioneProduzioneMacchina->macchina\_descrizione = $macchina[**'macchina\_descrizione'**];  
 $pianificazioneProduzioneMacchina->tipomacchina\_codice = $macchina[**'tipomacchina\_codice'**];  
 $pianificazioneProduzioneMacchina->tipomacchina\_descrizione = $macchina[**'tipomacchina\_descrizione'**];  
 $pianificazioneProduzioneMacchina->tempoutilizzo = $macchina[**'tempoutilizzo'**];  
 $pianificazioneProduzioneMacchina->pianprodtesta\_id = $pianificazioneId;  
 $pianificazioneProduzioneMacchina->save();  
  
 $pianificazioneMacchinaId = $pianificazioneProduzioneMacchina->id;  
  
 //creo un record tabella pianificazione\_produzione\_ordine per ogni ordine di produzione assegnato alla macchina  
 $sequenza = 1;  
 foreach ($macchina[**'ordiniproduzione'**] as $ordineProduzione) {  
 $pianificazioneProduzioneOrdine = new PianificazioneProduzioneOrdine();  
 $pianificazioneProduzioneOrdine->numeroordine = $ordineProduzione[**'numeroordine'**];  
 $pianificazioneProduzioneOrdine->prodotto\_codice = $ordineProduzione[**'prodotto\_codice'**];  
 $pianificazioneProduzioneOrdine->prodotto\_descrizione = $ordineProduzione[**'prodotto\_descrizione'**];  
 $pianificazioneProduzioneOrdine->quantita = $ordineProduzione[**'quantita'**];  
 $pianificazioneProduzioneOrdine->datascadenza = $ordineProduzione[**'datascadenza'**];  
 $pianificazioneProduzioneOrdine->datainizio = $ordineProduzione[**'datainizio'**];  
 $pianificazioneProduzioneOrdine->datafine = $ordineProduzione[**'datafine'**];  
 $pianificazioneProduzioneOrdine->pianprodmacchina\_id = $pianificazioneMacchinaId;  
 $pianificazioneProduzioneOrdine->sequenza = $sequenza;  
 $pianificazioneProduzioneOrdine->save();  
 $sequenza += 1;  
 }  
 }  
 }  
}

# **8. Struttura Progetto**

La struttura delle cartelle del progetto è quella fornita dal framework Laravel. Di seguito una breve spiegazione per orientarsi all’interno della struttura del progetto.



Nella repository principale del progetto oltre alle varie cartelle vi sono una serie di file utili per la configurazione e la gestione del progetto. In particolare:

**composer.json**: questo file contiene la configurazione per il tool Composer. Il file contiene una struttura JSON in grado di essere interpretata da Composer.

Composer è un tool per la gestione delle dipendenze di librerie PHP. Il funzionamento di composer è basato sulla presenza di un file di configurazione denominato composer.json. Questo file è utilizzato per impostare le dipendenze con librerie di terze parti da utilizzare nel progetto.

**packcage.json:** questo file contiene la configurazione per il tool NPM. Il file contiene una struttura JSON in grado di essere interpretata da NPM. NPM è un tool di package manager per la gestione delle dipendenze di libreria JavaScript. Quindi a differenza di Composer che si occupa delle librerie PHP per il backend dell’applicazione, NPM si occupa della gestione delle librerie JavaScript per il frontend dell’applicazione.

**phpunit.xml:** questo file contiene la configurazione per il tool PHPUnit utilizzato per i test di unità e di copertura.

**psalm.xml:** questo file contiene la configurazione per il tool Psalm utilizzato per l’analisi statica del codice.

**.env:** file contenente le variabili ambientali utilizzate all’interno del progetto.

**vendor:** cartella contenente tutte le librerie PHP necessarie nel progetto. Viene creata con l’utilizzo del tool Composer.

**node\_modules:** cartella contenente tutte le librerie JavaScript necessarie al frontend del progetto. Viene creata con l’utilizzo di NPM.

Nella cartella **app** sono presenti i file per l’implementazione del backend dell’applicazione.

Immagine che contiene testo

Descrizione generata automaticamente

Nella foto sono evidenziati i file implementati durante lo sviluppo dell’applicazione. All’interno di questi file vi è l’implementazione delle classi per la gestione dei componenti Model e Controller.

Il linguaggio utilizzato per l’implementazione di queste classi, come detto in precedenza, è il PHP.

Nelle cartelle **public** e **resources** sono contenuti invece tutti i file per l’implementazione del frontend dell’applicazione.

In particolare, nella repository **resoruces/views** sono presenti tutti i file per l’implementazione dei componenti View.

Questi file hanno estensione ‘blade.php’ perché implementate utilizzando il motore di template PHP di Laravel, **Blade.** I linguaggi utilizzati sono HTML, CSS e JavaScript.

Immagine che contiene testo

Descrizione generata automaticamente