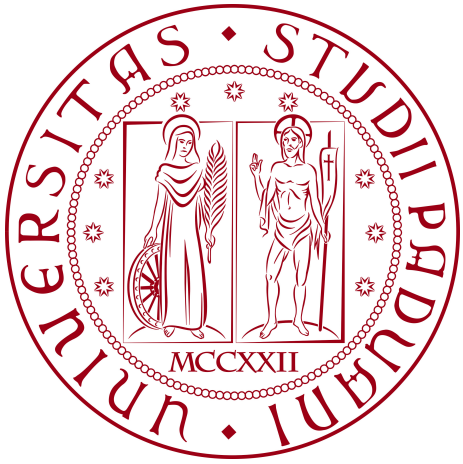
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Ottimizzazione della produzione in una sala stampa 3D: programmazione a vincoli e Google OR-Tools.

Tesi di laurea triennale

Relatrice

Prof.ssa Losiouk Eleonora

Laureando

Mazzotti Matteo

Matricola 2068245

Anno Accademico 2025-2026

Ringraziamenti

Desidero esprimere la mia gratitudine alla professoressa Losiouk Eleonoraper l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Vorrei anche ringraziare, con affetto, i miei genitori per il loro sostegno, il grande aiuto e la loro presenza in ogni momento durante gli anni di studio.

Desidero poi ringraziare i miei amici per i bellissimi anni trascorsi insieme e le mille avventure vissute.

Padova, Dicembre 2025 *Mazzotti Matteo*

Sommario

L'azienda Spazio Dev dispone di una sala stampa 3D la cui pianificazione manuale della produzione richiede un notevole tempo operativo. Il presente lavoro descrive lo sviluppo di un sistema automatizzato, basato sulla programmazione a vincoli e sull'utilizzo di Google OR-Tools (CP-SAT), in grado di proporre all'operatore una coda di stampa ottimizzata secondo le buone pratiche aziendali. Il sistema, integrato con il software gestionale esistente, costituisce una soluzione concreta al problema della pianificazione manuale, contribuendo a migliorare la produttività della sala stampa e a ottimizzare l'impiego delle risorse.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici sorgenti

Acronimi e abbreviazioni

ASGI Asynchronous Server Gateway Interface. 26

JWT JSON Web Token. 26

ORM Object-Relational Mapping. 25

Glossario

API In informatica un'Application Programming Interface (API) è un insieme di regole, contratti e punti di accesso che un componente software espone per consentire ad altri componenti o servizi di interagire in modo controllato, nascondendo i dettagli implementativi e semplificando l'integrazione tra sistemi eterogenei. 4

Backlog Elenco ordinato e dinamico di elementi di lavoro (item) che rappresentano valore da realizzare. In Scrum, il Product Backlog, di responsabilità del Product Owner, raccoglie e priorizza le esigenze del prodotto; lo Sprint Backlog è il piano dei Developer per raggiungere lo Sprint Goal durante lo Sprint. 7

Bugfix Modifica del codice, della configurazione o dei dati volta a rimuovere un malfunzionamento (bug) e a ripristinare il comportamento atteso del sistema senza introdurre cambiamenti funzionali non richiesti. 6

Bug In informatica, errore di funzionamento di un sistema o di un programma. 6

Casi d'uso Descrizioni strutturate di come gli attori interagiscono con il sistema per raggiungere obiettivi specifici, evidenziando flussi principali, varianti ed esigenze funzionali che guidano l'analisi dei requisiti. 15

Client Attore software o dispositivo che invia richieste a un servizio o *API* remoto utilizzando credenziali come `client_id` e `client_secret`; può rappresentare applicazioni, integrazioni di terze parti o componenti interne autorizzate a consumare l'interfaccia esposta. 27

Code style Insieme di regole e convenzioni formali che disciplinano formattazione, nomenclatura e organizzazione del codice sorgente, così da mantenerlo leggibile e uniforme all'interno di un team. 25

Daily Scrum Evento giornaliero, time-box di 15 minuti, in cui i Developer ispezionano i progressi verso lo Sprint Goal e adattano il piano per le prossime 24 ore; non è una riunione di status per gli stakeholder. 7

Dependency Injection Tecnica concreta che realizza l'inversione del controllo fornendo a un componente le istanze delle sue dipendenze attraverso costruttori, metodi o parametri, tipicamente orchestrata da un contenitore che risolve e configura gli oggetti richiesti. 30

Deploy Processo di rilascio e messa in esercizio di un'applicazione su un ambiente target (test, staging, produzione), comprendendo packaging, distribuzione, configurazione e attivazione dei servizi necessari per renderla disponibile agli utenti. 26

Design Pattern Soluzione progettuale riutilizzabile che descrive come risolvere un problema ricorrente di design software, codificando ruoli, responsabilità e interazioni tra componenti per migliorare manutenibilità, flessibilità e comunicazione all'interno del team. 23, 29

Design pattern architetturale Schema di alto livello che definisce composizione e interazione dei componenti di un sistema software per soddisfare requisiti non funzionali come scalabilità, resilienza o sicurezza, fornendo linee guida per strutturare l'intera architettura. 29

Feature Unità coerente di comportamento di un sistema che produce un beneficio osservabile per l'utente. 5

Framework Insieme coerente di componenti software riutilizzabili che fornisce struttura, astrazioni e convenzioni per sviluppare una specifica classe di applicazioni, velocizzando lo sviluppo e favorendo la consistenza del codice. 24

Funzione obiettivo Espressione che misura la qualità di una soluzione in un modello di ottimizzazione, da minimizzare o massimizzare nel rispetto dei vincoli definiti. 60

Git branching model Un modello di branching Git è una strategia o un insieme di regole che definisce come i team devono creare, gestire e unire i branch in un repository Git, al fine di organizzare il flusso di sviluppo. 5

Inversione del controllo Principio architetturale secondo cui un componente non istanzia né gestisce direttamente le proprie dipendenze ma delega a un contenitore o infrastruttura esterna la responsabilità di fornirle, riducendo l'accoppiamento e facilitando test e sostituzione delle implementazioni. 29

Issue Unità di lavoro o ticket che rappresenta una richiesta, un bug, un miglioramento o un'attività da tracciare e gestire in un sistema di gestione del lavoro come Jira o GitHub Issues. Ogni issue dovrebbe includere contesto, criteri di accettazione e stato per facilitare la collaborazione. 8

Layer Strato logico dell'architettura che incapsula un insieme coerente di responsabilità e offre servizi attraverso interfacce ben definite agli strati adiacenti, limitando l'accoppiamento e favorendo la separazione delle preoccupazioni. 29

Logica di business Insieme delle regole applicative e delle trasformazioni sui dati che implementano i requisiti del dominio, orchestrando servizi, repository e validazioni per garantire coerenza funzionale indipendentemente da presentazione e persistenza. 34

Makespan Intervallo temporale complessivo necessario per completare tutte le attività pianificate, pari alla differenza tra l'avvio del primo lavoro e la conclusione dell'ultimo all'interno della schedulazione. 40

Multi tenant Modello architetturale in cui un'unica istanza dell'applicazione serve più tenant isolati (organizzazioni o gruppi di utenti) condividendo

infrastruttura e codice ma mantenendo separati dati e configurazioni, così da ottimizzare i costi e semplificare il rilascio. 28

Open source Modello di distribuzione del software in cui il codice sorgente è reso pubblico con una licenza che consente a chiunque di studiarlo, modificarlo e ridistribuirlo, favorendo collaborazione e trasparenza. 26

Ottimizzazione combinatoria Area dell'ottimizzazione che studia problemi in cui si cerca la soluzione migliore tra un insieme finito ma molto ampio di combinazioni discrete, tipicamente soggetti a vincoli (es. scheduling, routing, assegnamento). 26

Pattern a layer Design pattern architetturale che suddivide un sistema in strati indipendenti con responsabilità specifiche (es. presentazione, logica, accesso ai dati), così da ridurre le dipendenze e facilitare riuso, manutenzione e test. 29

POC Il Proof Of Concept (POC) è l'allestimento di una demo prototipale del sistema o applicazione in sviluppo o in corso di valutazione. 3

Product Owner Figura chiave di Scrum responsabile di massimizzare il valore del prodotto e del lavoro del team. Definisce e mantiene il Product Backlog, ne ordina gli elementi in base al valore e agli obiettivi, chiarisce i requisiti e accetta l'incremento completato. 7

Scalabilità Capacità di un sistema di sostenere carichi crescenti (utenti, dati o richieste) mantenendo livelli di servizio accettabili, adattando risorse hardware o struttura software tramite scaling verticale, orizzontale o elastico. 29

Scrum Master Servant leader del team Scrum. Promuove e supporta Scrum come definito nella Scrum Guide, facilita gli eventi Scrum, rimuove impedimenti, tutela il team e aiuta l'organizzazione ad adottare pratiche agili. 7

Server di deploy Computer dedicato all'esecuzione in produzione di un'applicazione software. Ospita il codice distribuito, espone le relative API e garantisce risorse hardware e configurazioni adeguate per gestire il carico operativo previsto.. 13

Sprint Time-box tipico del framework Scrum, della durata compresa fra uno e quattro settimane, durante il quale il team realizza un incremento potenzialmente rilasciabile seguendo uno Sprint Goal condiviso. Lo Sprint include pianificazione, sviluppo, revisione e retrospettiva. 8

Sprint Planning Evento che apre lo Sprint in cui lo Scrum Team definisce lo Sprint Goal, seleziona gli elementi del Product Backlog da includere nello Sprint e pianifica il lavoro necessario nel Sprint Backlog. 7

Sprint Retrospective Ultimo evento dello Sprint dedicato all'ispezione delle modalità di lavoro del team e all'individuazione di miglioramenti pratici da implementare nel prossimo Sprint. 7

Sprint Review Evento alla fine dello Sprint per ispezionare l'incremento e adattare il Product Backlog. Coinvolge stakeholder e team per raccogliere feedback, rivedere i risultati e allineare i prossimi passi. 7

Test di integrazione Caso di test che verifica l'interazione corretta tra più componenti o sottosistemi (ad esempio modulo di schedulazione, accesso al database e servizi esterni), evidenziando problemi dovuti all'integrazione piuttosto che alla singola unità di codice. 25

Test di unità Caso di test automatizzato che verifica in isolamento il comportamento di una singola unità di codice (ad esempio una funzione, un metodo o una classe), sostituendo le dipendenze esterne con stub o mock per concentrarsi esclusivamente sulla logica interna. 25

Test funzionale Caso di test che verifica il comportamento osservabile del sistema rispetto ai requisiti funzionali, esercitando un flusso completo di utilizzo (ad esempio uno scenario di business) senza necessariamente coinvolgere tutti i componenti esterni reali. 25

Version Control System Un sistema di versionamento (Version Control System o VCS) è uno strumento software che traccia e gestisce le modifiche apportate a un file o a un insieme di file nel tempo, permettendo di recuperare versioni precedenti e di collaborare con altri utenti. 5

Capitolo 1

Introduzione

1.1 L'azienda

Spazio Dev S.r.l.[1] è una software house situata a Tombolo (PD) fondata da due soci, ad oggi l'azienda conta circa 17 dipendenti e si occupa di offrire servizi legati al mondo dello sviluppo web e dell'ottimizzazione dei processi industriali. L'azienda si occupa, nello specifico, di sviluppare siti web, e-commerce, software su misura e di integrare algoritmi che sfruttano l'intelligenza artificiale per ottimizzare la produzione o monitorare lo stato dell'azienda in tempo reale. Il logo aziendale è mostrato nella figura 1.1.

Figura 1.1: Logo dell'azienda Spazio Dev

1.1.1 Mugalab

Mugalab[2], il cui logo è mostrato nella figura 1.2, è la divisione di Spazio Dev S.r.l. dedicata alla gestione della sala stampa 3D presente nella sede aziendale. Attraverso la tecnica della stampa additiva, Mugalab si occupa di:

- progettare e prototipare componenti per il settore industriale;

[1] *Sito Spazio Dev.* URL: https://spaziodev.eu/.

[2] *Sito Mugalab.* URL: https://mugalab.com/.

- progettare e prototipare componenti per dispositivi elettronici;
- disegnare e produrre oggetti ornamentali.

Figura 1.2: Logo di Mugalab

1.2 Il progetto

1.2.1 L'idea

L'idea di questo percorso di stage curricolare nasce dalla necessità dell'azienda di gestire in maniera efficiente la produzione delle componenti stampate in 3D. Attualmente lo stabilimento dispone di 12 stampanti e la pianificazione della stampa degli ordini di produzione viene effettuata manualmente da un singolo operatore, il quale si accerta di ottimizzare la produzione seguendo alcune buone pratiche che permettono di risparmiare tempo e aumentarne l'efficienza. L'obiettivo principale dell'azienda è quindi quello di ideare un sistema automatizzato, il cui scopo è creare una coda di stampa ottimizzata a partire dagli ordini di produzione non ancora evasi completamente. Tale sistema dovrà integrarsi completamente con il sistema gestionale esistente, il quale viene utilizzato quotidianamente dal personale dell'azienda per gestire la coda di stampa e registrare gli ordini ricevuti dalle piattaforme utilizzate per la vendita delle componenti e degli oggetti ornamentali (Amazon, Shopify e vendita B2B). Questo progetto è destinato esclusivamente ad uso interno dell'azienda, assumendo quindi il ruolo

di *Proof Of Concept*$_G$ della fattibilità di un sistema di schedulazione automatizzato e vincolato. Il codice potrà quindi essere ulteriormente ottimizzato e ampliato in modo tale da migliorarne l'efficienza e aumentare la qualità del risultato prodotto, oltre ad essere adattato a scenari d'utilizzo differenti da quello proposto.

1.2.2 Motivazioni e problematiche attuali

La pianificazione manuale della coda di stampa genera ritardi e inefficienze operative, con ricadute dirette sui tempi di consegna. Una schedulazione basata su un algoritmo di ottimizzazione consente di generare piani fattibili in modo più rapido e meno incline a errori, migliorando l'utilizzo delle stampanti e riducendo i tempi morti.

1.3 Organizzazione del testo

Il secondo capitolo descrive il contesto aziendale, i processi di sviluppo software e la gestione della sala stampe.

Il terzo capitolo descrive il progetto di stage, l'analisi preventiva dei rischi e la pianificazione.

Il quarto capitolo contiene l'analisi dei requisiti del progetto.

Il quinto capitolo descrive la fase di codifica, progettazione e validazione del prodotto.

Il sesto capitolo contiene un'analisi critica del prodotto.

Durante la stesura del testo sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;

- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *Application Program Interface*$_G$;
- i nomi degli oggetti, delle funzioni, delle tabelle, delle colonne, delle classi e delle rotte *API* sono evidenziati con il carattere `monospaziato`.
- i termini in lingua straniera o facenti parte del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Il contesto aziendale

2.1 Sviluppo software

Spazio Dev è un'azienda di piccole dimensioni, di recente fondazione e ancora in fase di crescita. Di conseguenza, il progetto è stato sviluppato con budget limitato e da una singola persona. I processi di sviluppo aziendali, pertanto, sono ancora in rapida evoluzione.

2.1.1 Version Control System

L'azienda utilizza Git[1] come *Version Control System*$_G$ e Gitea[2] come piattaforma di hosting per i repository. Per quanto riguarda il *Git branching model*$_G$ viene adottato il Gitflow workflow[3], adattato nel seguente modo:

- il *branch* dedicato alle versioni stabili è denominato *main*;
- il *branch* dedicato alle versioni di sviluppo è denominato *develop*;
- i *branch* dedicati alle implementazioni di *feature*$_G$ vengono denominati con lo schema *feature/<nome_feature>* dove il parametro *<nome_feature>* è una breve descrizione della *feature* che si sta codificando;

[1] *Sito ufficiale Git.* URL: https://git-scm.com/.

[2] *Sito ufficiale Gitea.* URL: https://about.gitea.com/.

[3] *Spiegazione completa Gitflow workflow.* URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

- i *branch* dedicati ai *bugfix*$_G$ vengono denominati con lo schema *fix/<nome_bug>* dove il parametro *<nome_bug>* è una breve descrizione del *bug*$_G$ che si sta risolvendo.

- i *branch* dedicati alla documentazione vengono denominati con lo schema *docs/<nome_documentazione>* dove il parametro *<nome_documentazione>* è una breve descrizione della modifica alla documentazione.

Il funzionamento del Gitflow Workflow viene mostrato nella figura 2.1.

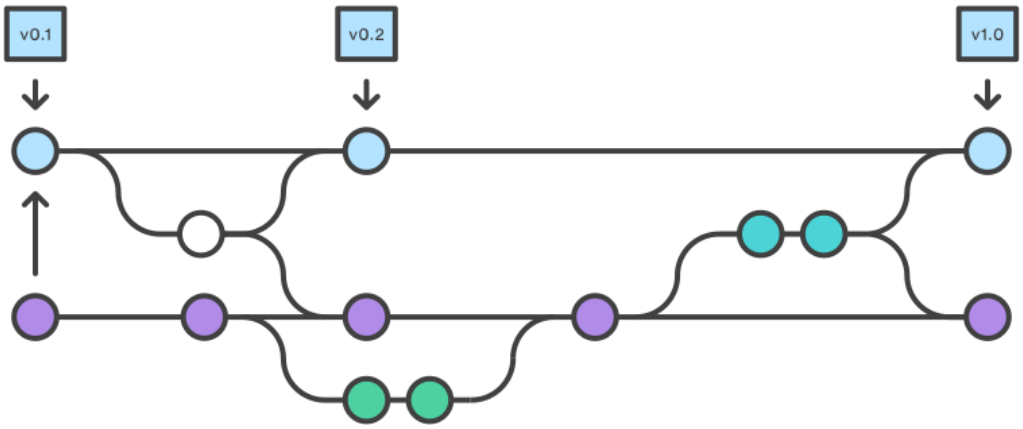


Figura 2.1: Funzionamento Gitflow Workflow

Viene inoltre adottato lo standard Conventional Commits 1.0.0[4] per la scrittura dei messaggi di commit, in particolare ogni commit possiede un messaggio così formato: *<tipo>: <descrizione>*, dove il parametro *<tipo>* è:

- *fix* in caso di commit contenente *bugfix*;

- *docs* in caso di commit di aggiornamento della documentazione;

- *feat* in caso di commit contenente *feature*.

Non sono state definite policy di protezione dei rami, requisiti di revisione o regole di merge, tali aspetti potranno tuttavia essere definiti in futuro dall'azienda.

[4] *Documentazione ufficiale Conventional Commits 1.0.0.* URL: https://www.conventionalcommits.org/en/v1.0.0/.

2.1.2 Modello di lavoro agile

L'azienda adotta il framework Scrum[5] come modello di lavoro agile[6], in particolare:

- il ruolo di *Product Owner*$_G$ viene ricoperto dai titolari dell'azienda i quali si occupano di creare e gestire il *backlog*$_G$ del prodotto, fornire indicazioni al team su quali feature implementare e decidono le scadenze dei rilasci del prodotto;
- il ruolo di *Scrum Master*$_G$ viene ricoperto dai titolari dell'azienda, i quali si occupano di pianificare *Sprint Planning*$_G$, *Daily Scrum*$_G$, *Sprint Review*$_G$ e *Sprint Retrospective*$_G$;
- il ruolo del team di sviluppo viene ricoperto dagli sviluppatori, i quali vengono divisi in gruppi e assegnati ai progetti in fase di sviluppo.

Il funzionamento del framework Scrum viene mostrato nella figura 2.1.

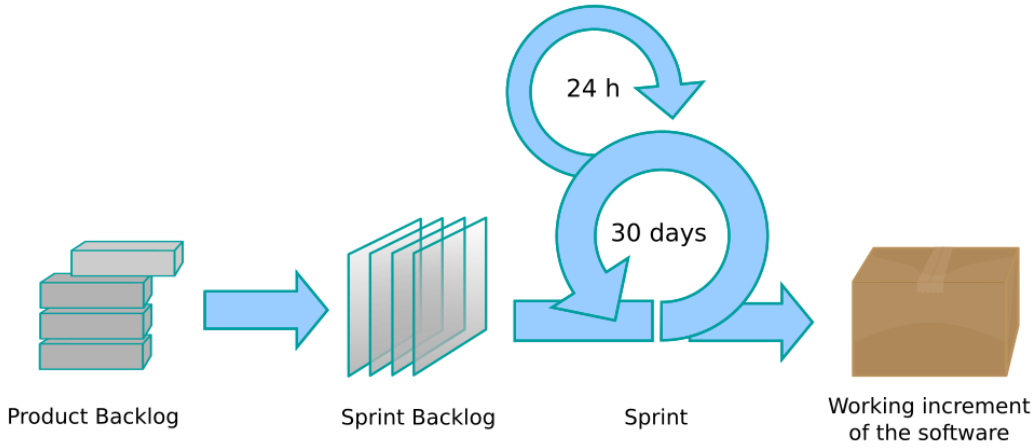


Figura 2.2: Framework Scrum

[5] *Framework Scrum*. URL: https://it.wikipedia.org/wiki/Scrum_(informatica).

[6] *Manifesto Agile*. URL: http://agilemanifesto.org/iso/it/.

2.1.3 Issue Tracking System

L'azienda utilizza Plane[7] come sistema di tracciamento delle attività. Le *issue*$_G$ sono organizzate su una board in stile Kanban composta da cinque colonne:

- *Backlog*: raccolta iniziale delle attività proposte, non ancora pianificate;
- *Todo*: attività prioritarie selezionate per l'esecuzione;
- *In Progress*: attività in lavorazione;
- *In Review*: attività concluse e in fase di verifica;
- *Done*: attività completate.

Ogni *issue* è inoltre associata a un livello di priorità tra i seguenti:

- *Critical*: attività richiedenti intervento immediato;
- *High*: richieste ad alto impatto o con scadenza molto ravvicinata;
- *Medium*: *bugfix* non critici che vengono inseriti nella pianificazione ordinaria degli *Sprint*$_G$;
- *Low*: miglioramenti minori non prioritari.

La combinazione di colonna Kanban e priorità rende evidente sia lo stato di avanzamento sia l'urgenza. La struttura della Kanban board è mostrata nella figura 2.3.

[7] *Sito ufficiale Plane*. URL: https://plane.so/.

Figura 2.3: Kanban board

2.1.4 Strumenti di comunicazione

Le comunicazioni interne avvengono interamente tramite Telegram[8]: ogni progetto dispone di un gruppo dedicato in cui titolari e sviluppatori condividono aggiornamenti, documentazione e decisioni operative in tempo reale. Questo canale unico permette di evitare dispersioni e di mantenere traccia delle richieste provenienti dai clienti.

Le riunioni operative vengono organizzate con cadenza variabile, in funzione delle necessità del progetto o delle scadenze concordate con i clienti. Quando emergono nuove priorità o si avvicina un rilascio, i titolari convocano incontri mirati per allineare il team sugli obiettivi, discutere eventuali impedimenti e definire le attività da pianificare nel successivo *sprint*.

2.2 Gestione della sala stampe

Spazio Dev possiede attualmente molteplici punti di ingresso degli ordini relativi alla stampa 3D dei componenti, in particolare:

[8] *Telegram*. Sito ufficiale Telegram. URL: https://telegram.org (visitato il giorno 23/10/2025).

- La vendita di vasi portafiori e complementi d'arredo viene effettuata online e sulla piattaforma e-commerce Amazon;
- Le produzioni personalizzate e B2B vengono gestite manualmente.

Per la gestione di tali ordini e della coda di stampa viene utilizzato un gestionale sviluppato dall'azienda stessa, denominato "Idrotech Manager".

2.2.1 Ottimizzazione della coda di stampa

Per gestire in maniera efficiente l'evasione degli ordini l'azienda applica diverse ottimizzazioni alla coda di stampa, in modo tale da sfruttare nel miglior modo possibile le capacità di produzione. Tali procedure riguardano in particolare gli orari di avvio della stampa, la gestione dei cambi di materiale e dei cambi di ugello.

2.2.1.1 Orari di avvio della stampa

La gestione degli orari di avvio della stampa risulta fondamentale per ottimizzare al meglio la produzione. Ciò è dovuto al fatto che ogni stampante, quando termina la produzione di un piatto, necessita di un intervento manuale da parte di un operatore per la rimozione del prodotto finito e l'avvio della stampa successiva. L'idea alla base è quindi quella di pianificare tutte le stampe brevi durante l'orario lavorativo (in modo che sia sempre possibile avviare una nuova produzione). Le stampe più lunghe vengono invece avviate poco prima della fine della giornata per sfruttare al meglio i periodi di tempo in cui non c'è personale presente all'interno dello stabilimento.

2.2.1.2 Cambi di materiale

La coda di stampa può essere ottimizzata riducendo al minimo i cambi di materiale (o colore) tra una stampa e quella successiva. Questo perché ogni qual volta viene richiesto di cambiare il colore del filamento o il materiale utilizzato la stampa viene interrotta e un operatore deve intervenire manualmente per la sostituzione del filamento. Per semplicità, l'azienda assume che il tempo necessario per cambiare un filamento sia di 5 minuti.

2.2.1.3 Cambi ugello

Ogni articolo possiede uno specifico ugello che viene utilizzato per cambiare lo spessore della stampa. Come per i cambi di materiale anche il cambio ugello richiede un intervento manuale da parte di un operatore. Per semplicità, l'azienda assume che il tempo necessario per cambiare un ugello sia di 15 minuti.

Capitolo 3

Descrizione dello stage

3.1 Introduzione al progetto

Come descritto nell'introduzione (1) il progetto codificato durante il periodo di stage curricolare è uno schedulatore automatico che ottimizza e riordina la coda di stampa a partire dagli ordini di produzione presenti all'interno del software gestionale Idrotech Manager. Tale schedulatore deve tenere conto delle ottimizzazioni descritte nella sezione dedicata (2.2), oltre a doversi integrare con il gestionale in utilizzo dal personale.

3.2 Analisi preventiva dei rischi

La prima fase dello stage curricolare è stata dedicata all'analisi preventiva dei rischi. Tale procedura ha lo scopo di delineare i possibili rischi a cui si può andare incontro durante lo sviluppo del progetto, oltre a trovare delle possibili soluzioni per mitigare tali problematiche.

1. Errata scelta delle tecnologie

Descrizione: una scelta errata delle tecnologie da utilizzare per la codifica del progetto potrebbe portare ad un risultato inutilizzabile o non abbastanza performante.

Soluzione: coinvolgimento del responsabile nella scelta e occupare il primo periodo per validare le idee proposte.

2. Sicurezza nell'integrazione del modulo

Descrizione: la comunicazione tra il modulo da sviluppare e il gestionale esistente deve rispettare degli standard minimi di sicurezza, in modo che non tutti possano accedere alle *API* esposte.

Soluzione: dedicare parte del tempo allo studio su come rendere più sicuro l'accesso alle rotte *API*.

3. Risultati di ordinamento insoddisfacenti

Descrizione: il modulo potrebbe restituire risultati non soddisfacenti, con conseguenti inefficienze nella produzione.

Soluzione: validare assieme al tutor interno le logiche implementate in maniera periodica, testare il modulo con dataset realistici.

4. Prestazioni dello schedulatore

Descrizione: il modulo potrebbe impiegare troppo tempo per ottimizzare la coda di stampa.

Soluzione: utilizzare un *server di deploy*$_G$ con buone prestazioni, velocizzare l'esecuzione diminuendo, ad esempio, il tempo disponibile per la ricerca di una soluzione ottima.

5. Rispetto delle scadenze

Descrizione: le tempistiche dello stage curricolare potrebbero non essere sufficienti per avere un risultato concreto e utilizzabile.

Soluzione: svolgere una pianificazione (3.3) accurata del periodo di tempo a disposizione e convalidare il raggiungimento degli obiettivi.

3.3 Pianificazione

Oltre all'analisi preventiva dei rischi (3.2) è stata svolta una pianificazione accurata di tutte le fasi del periodo di stage. Questo, oltre a mitigare il rischio "Rispetto delle scadenze" (5), serve a determinare i contenuti da revisionare al termine di ogni *sprint*.

Settimana	**Obiettivi**	**Ore**
1	Analisi dei requisiti e dei rischi del progetto, scelta e inizio studio delle tecnologie.	40
2	Lettura della documentazione, studio del gestionale e progettazione delle integrazioni.	40
3	Inizio della codifica del progetto.	40
4	Codifica del progetto.	40
5	Codifica del progetto.	40
6	Codifica del progetto.	40
7	Codifica del progetto, validazione logiche implementate.	40
8	Test dell'algoritmo con dataset realistici.	20

Tabella 3.1: Tabella riassuntiva della pianificazione del periodo di stage.

Capitolo 4

Analisi dei requisiti

4.1 Casi d'uso

Come descritto nella sezione Pianificazione la prima parte del periodo di stage è stata dedicata all'analisi dei requisiti del progetto. Prima di fare ciò sono stati definiti tutti i *casi d'uso*$_G$ del sistema.

4.1.1 Attori individuati

L'utente finale del sistema non utilizzerà direttamente il modulo di pianificazione, le sue funzionalità potranno essere messe a disposizione attraverso il software gestionale, per questo sono stati individuati due principali attori:

- operatore, ovvero l'utente che interagisce con il software gestionale;
- software gestionale, il quale interagisce con il modulo di pianificazione.

Trattandosi di un sistema con il compito di automatizzare un processo, il numero dei *casi d'uso* risulta limitato.

4.1.2 Tabella dei casi d'uso

Identificativo	**Descrizione**
UC1	*Attori*: Gestionale. *Scopo*: Ricevere una lista di ordini da pianificare dal software gestionale. *Pre-condizione*: Endpoint raggiungibile, autenticazione valida e payload validato correttamente. *Post-condizione*: Richiesta accettata e messa in coda per l'elaborazione.
UC1.1	*Attori*: Gestionale. *Scopo*: Calcolare un piano di stampa coerente con i vincoli. *Pre-condizione*: Esiste una richiesta di schedulazione accettata. *Post-condizione*: Piano di schedulazione prodotto e pronto all'invio.
UC1.2	*Attori*: Gestionale. *Scopo*: Notificare al gestionale l'esito della schedulazione. *Pre-condizione*: Risultato disponibile, webhook del gestionale configurato e raggiungibile. *Post-condizione*: Esito notificato e inviato al gestionale.

Continua...

Continua...

Identificativo	**Descrizione**
UC1.E	*Attori*: Gestionale. *Scopo*: Comunicare un errore occorso in UC1 o UC1.1. *Pre-condizione*: Si è verificata un'anomalia (mancata autorizzazione, validazione dei dati fallita o altro). *Post-condizione*: Errore tracciato e inviato al gestionale.
UC2	*Attori*: Operatore. *Scopo*: Consultare l'elenco dei risultati di schedulazione. *Pre-condizione*: Schedulazioni precedenti presenti. *Post-condizione*: Schedulazioni mostrate.
UC2.1	*Attori*: Operatore. *Scopo*: Visualizzare il piano su vista a calendario. *Pre-condizione*: Almeno una schedulazione presente nel database. *Post-condizione*: Calendario visualizzabile a schermo con gli slot pianificati.

Continua...

Continua...

Identificativo	**Descrizione**
UC2.2	*Attori*: Operatore. *Scopo*: Confermare l'accettazione della coda proposta. *Pre-condizione*: Sezione dei dettagli della schedulazione aperta. *Post-condizione*: Schedulazione marcata come accettata e importazione avvenuta nel calendario.
UC2.E	*Attori*: Operatore. *Scopo*: Informare l'utente di errori avvenuti durante l'importazione a calendario. *Pre-condizione*: Errore occorso in UC2.2. *Post-condizione*: Errore comunicato e stato invariato.
UC3	*Attori*: Operatore. *Scopo*: Richiedere l'avvio della schedulazione automatica. *Pre-condizione*: Operatore autenticato, sezione Ordini accessibile. *Post-condizione*: Richiesta inviata al modulo e accettata.

Continua...

Continua...

Identificativo	Descrizione
UC3.1	*Attori*: Operatore. *Scopo*: Selezionare uno o più ordini da includere nella schedulazione. *Pre-condizione*: Modale di selezione ordini aperto. *Post-condizione*: Ordini da pianificare selezionati e pronti all'invio.
UC3.2	*Attori*: Operatore. *Scopo*: Informare dell'avvenuta accettazione della richiesta da parte del modulo. *Pre-condizione*: Risposta positiva del modulo. *Post-condizione*: Notifica visualizzata, modale chiuso.
UC3.E	*Attori*: Operatore *Scopo*: Informare l'utente di errori incontrati durante l'avvio della schedulazione. *Pre-condizione*: Errore occorso in UC3. *Post-condizione*: Errore comunicato e stato invariato.

4.1.3 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono riportati nelle figure 4.1, 4.2 e 4.3.

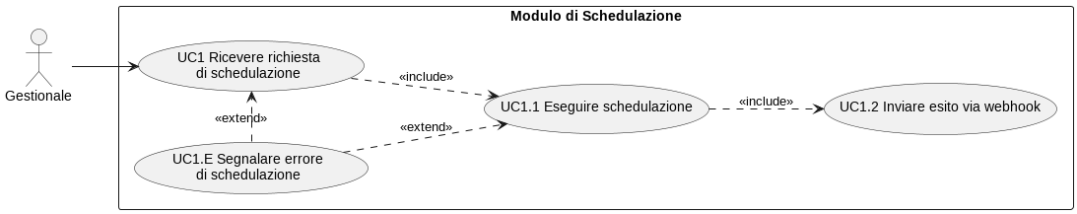


Figura 4.1: Diagramma dei casi d'uso UC1, UC1.1, UC1.2, UC1.E

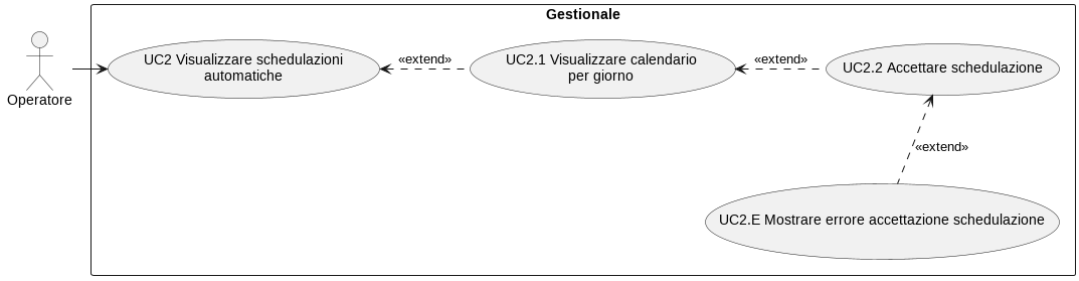


Figura 4.2: Diagramma dei casi d'uso UC2, UC2.1, UC2.2, UC2.E

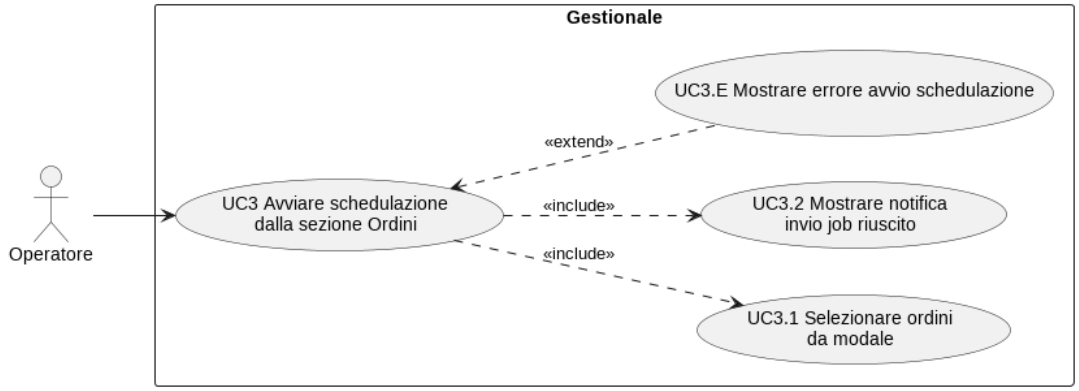


Figura 4.3: Diagramma dei casi d'uso UC3, UC3.1, UC3.2, UC3.E

4.2 Analisi dei requisiti

Una volta stilata la lista dei *casi d'uso* è stato possibile procedere con l'effettiva analisi dei requisiti del sistema. Tale operazione si è svolta attraverso un incontro con il tutor aziendale, durante il quale sono stati rilevati tutti i requisiti obbligatori, desiderabili e facoltativi.

4.2.1 Tracciamento dei requisiti

Ogni requisito viene identificato attraverso un codice identificativo di questo tipo:

R.x.y

dove:

- la lettera R identifica il requisito;

- la lettera x identifica la tipologia di tale requisito, ovvero:
  - O obbligatorio;
  - D desiderabile;
  - F facoltativo.
- la lettera y è un valore numerico progressivo a partire da 0.

4.3 Tabelle dei requisiti

4.3.1 Tabella dei requisiti obbligatori

Requisito	Descrizione
RO0	Il modulo deve produrre una coda di stampa dettagliata, evidenziando quale articolo stampare e quale stampante usare.
RO1	La schedulazione proposta deve considerare gli orari di lavoro aziendali.
RO2	La schedulazione proposta deve ottimizzare la stampa raggruppando i lavori con lo stesso materiale.
RO3	La schedulazione proposta deve ottimizzare la stampa pianificando i lavori corti durante la giornata lavorativa.
RO4	La schedulazione proposta deve ottimizzare la stampa pianificando i lavori lunghi al di fuori della giornata lavorativa.
RO5	La schedulazione deve essere avviabile dal software gestionale esistente.
RO6	La schedulazione deve essere visualizzabile dal software gestionale.
RO7	La schedulazione deve essere importabile nel calendario del software gestionale.
RO8	La schedulazione deve essere visibile in maniera chiara, in modo che l'utente possa trovare facilmente le informazioni utili.
RO9	La soluzione deve essere ammissibile e non devono esserci lavori sovrapposti.
RO10	Tutti i pezzi da stampare devono essere inseriti nella pianificazione, non sono ammesse stampe parziali.
RO11	Deve essere considerato un tempo di cambio piatto tra una stampa e l'altra.
RO12	Deve essere considerato un tempo di cambio filamento tra un lavoro di stampa e il suo successivo se usano materiali diversi.

Tabella 4.1: Tabella del tracciamento dei requisiti obbligatori.

4.3.2 Tabella dei requisiti desiderabili

Requisito	Descrizione
RD0	Deve essere disponibile un metodo di autenticazione per validare l'origine della richiesta delle schedulazioni.
RD1	Lo sviluppo deve avvenire utilizzando *design pattern*$_G$ comprovati e producendo codice mantenibile per facilitare le modifiche future.
RD2	La schedulazione proposta deve ottimizzare la stampa raggruppando i lavori con lo stesso ugello.
RD3	La schedulazione proposta deve ottimizzare la stampa raggruppando i lavori con lo stesso colore.
RD4	Al termine dello sviluppo l'algoritmo implementato deve essere validato utilizzando set di dati realistici.
RD5	La schedulazione proposta deve ottimizzare il numero di pezzi stampati nel piatto, minimizzando gli sprechi e il ritardo nelle consegne.

Tabella 4.2: Tabella del tracciamento dei requisiti desiderabili.

4.3.3 Tabella dei requisiti facoltativi

Requisito	Descrizione
RF0	La pianificazione deve avvenire automaticamente ad intervalli di tempo prefissati e modificabili dall'utente.
RF1	La pianificazione deve poter essere calcolata a partire da uno stato iniziale, schedulando solo i lavori mancanti.
RF2	L'utente deve essere in grado di modificare i parametri dello schedulatore dal software gestionale.

Tabella 4.3: Tabella del tracciamento dei requisiti facoltativi.

Capitolo 5

Progettazione e codifica

La fase di progettazione e codifica è stata la più significativa per il progetto in quanto ha occupato la maggior parte del tempo a disposizione. Attraverso un incontro con il tutor aziendale sono stati definiti i *framework*$_G$ per la codifica e l'architettura del sistema.

5.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

5.1.1 Linguaggi e framework

5.1.1.1 Python

Linguaggio di programmazione ad alto livello, orientato agli oggetti e adatto a sviluppare applicazioni distribuite. Viene utilizzato nello sviluppo web, nell'analisi dei dati, nel machine learning e nello sviluppo di automazioni. Permette di installare e gestire le librerie attraverso pip, un gestore di pacchetti.

5.1.1.2 FastAPI

Framework web per la codifica di *API* basato su Python[1]. Offre elevate prestazioni, validazione dell'input attraverso l'utilizzo della libreria Pydantic[2] e documentazione automatica seguendo lo standard OpenAPI[3].

5.1.1.3 Pytest

Framework che permette la scrittura e l'esecuzione di *test di unità*$_G$, *test funzionale*$_G$ e *test di integrazione*$_G$.

5.1.2 Librerie

5.1.2.1 Black

Libreria che permette di formattare il codice Python, utilizzata per imporre un *code style*$_G$ unico e ripetibile.

5.1.2.2 isort

Libreria che permette di formattare le importazioni delle librerie Python, utilizzata per imporre un *codestyle* unico e ripetibile.

5.1.2.3 SQLModel

Libreria che permette di interagire con basi di dati relazionali attraverso il codice scritto in Python con la tecnica di programmazione *Object-Relational Mapping (ORM)*$_G$.

[1] *Python*. Sito ufficiale Python. URL: https://www.python.org/ (visitato il giorno 08/11/2025).

[2] *Pydantic*. Sito ufficiale Pydantic. URL: https://docs.pydantic.dev/latest/ (visitato il giorno 08/11/2025).

[3] *OpenAPI*. Sito ufficiale documentazione OpenAPI. URL: https://swagger.io/specification/ (visitato il giorno 08/11/2025).

5.1.2.4 PyJWT

Libreria che permette di codificare e decodificare *JSON Web Token (JWT)*$_G$, utilizzati per autenticare utenti o servizi.

5.1.2.5 Pydantic

Libreria che aggiunge funzionalità legate alla validazione dei dati in Python, permette di definire con precisione la struttura dei dati e gestire i casi in cui si riceva un input non conforme alle aspettative.

5.1.2.6 Google OR-Tools

Libreria che contiene strumenti per risolvere problemi di *ottimizzazione combinatoria*$_G$. Permette di modellare un problema attraverso diversi linguaggi di programmazione (C++, Python, C# o Java) e utilizzare diversi risolutori commerciali o *open source*$_G$.

5.1.3 DevOps e server di deploy

5.1.3.1 Docker

Piattaforma utilizzata per creare ambienti isolati e riproducibili dove eseguire le applicazioni. Consente di standardizzare lo sviluppo locale e il *deploy*$_G$, riducendo differenze tra ambienti e semplificando l'esecuzione dei servizi.

5.1.3.2 Uvicorn

Server di tipo *Asynchronous Server Gateway Interface (ASGI)*$_G$ per applicazioni web Python asincrone. Utilizzato per eseguire l'*API* in sviluppo e in produzione, supporta la concorrenza.

5.1.4 Persistenza dei dati

5.1.4.1 MySQL

Sistema di gestione di basi di dati relazionali usato per la persistenza. Supporta transazioni e indici, ed è integrato con l'applicazione tramite *ORM* per la definizione dello schema e l'accesso ai dati.

5.2 Progettazione della base di dati

5.2.1 Base di dati del modulo di pianificazione

Assieme al tutor aziendale è stata presa la decisione di non utilizzare la base di dati del modulo di pianificazione per la persistenza dei risultati generati, questo perché la coda di stampa viene visualizzata direttamente all'interno del software gestionale e risulta pertanto più semplice e intuitivo modificare direttamente il *database* di quest'ultimo. La base di dati del modulo di pianificazione ha il solo compito di memorizzare i *client*$_G$ registrati e autorizzati a chiamare le *API* esposte, a tale scopo è stata creata una sola tabella denominata `clients` e così strutturata:

- `id`: intero auto-incrementale utilizzato come chiave primaria, generato automaticamente dal *database*.
- `client_id`: stringa obbligatoria indicizzata e univoca che identifica il client registrato.
- `client_secret`: stringa obbligatoria che contiene la password utilizzata dal client per l'autenticazione (vedi 5.8.2).
- `created_at`: campo di tipo `DateTime` per tracciare la data di registrazione.
- `is_active`: valore booleano che indica se il client è abilitato, è valorizzato a *true* di default.

Ciò permette, in futuro, di aggiungere eventuali altri *client* autorizzati a utilizzare il modulo.

5.2.2 Base di dati del software gestionale

La progettazione della base di dati è risultata fondamentale per permettere all'utente di visualizzare correttamente le code di stampa pianificate. Il software gestionale esistente (e già sviluppato al momento della codifica del progetto) possiede una base di dati già strutturata e utilizzata sia dall'azienda ospitante che da alcuni clienti dell'azienda stessa (in quanto il software è *multi tenant*$_G$). Per questo motivo l'ampliamento della base di dati esistente è stato eseguito cercando di mantenere al minimo il numero di modifiche.

Sono state aggiunte due tabelle per tracciare gli output della pianificazione e il legame con gli ordini:

- La tabella `scheduling_results` così formata:
  - `id`: intero auto-incrementale usato come chiave primaria;
  - `user_id`: chiave esterna verso la tabella `users` contenente i dati degli utenti registrati;
  - `data`: campo `json` obbligatorio che memorizza il payload completo della soluzione;
  - `is_accepted`: boolean obbligatorio valorizzato di default a *false* per indicare l'approvazione dell'utente;
  - `created_at`: campo di tipo `DateTime` per tracciare la data di ricezione del risultato;
  - `updated_at`: campo di tipo `DateTime` per tracciare la data dell'ultima modifica del risultato.
- La tabella `scheduling_results_orders` così formata:
  - `id`: intero auto-incrementale usato come chiave primaria;
  - `order_id`: chiave esterna verso la tabella `orders` contenente gli ordini ricevuti;

  - `scheduling_result_id`: chiave esterna verso `scheduling_results`.

5.3 Design pattern adottati

Qui vengono descritti i *design pattern*$_G$ comportamentali e di *inversione del controllo*$_G$ adottati.

5.3.1 Architettura del sistema

Il *design pattern architetturale*$_G$ scelto per la codifica del progetto è il *pattern a layer*$_G$. Tale scelta è stata presa per le seguenti motivazioni:

- Permette una buona separazione delle responsabilità, ogni *layer*$_G$ non conosce infatti l'implementazione degli altri;
- Permette una maggior semplicità di sviluppo, in quanto è un *design pattern architetturale* conosciuto e di semplice comprensione;
- Permette una buona manutenibilità del codice, ogni *layer* non viene influenzato dai cambiamenti apportati ad altri layer (a patto che i *contratti* tra i layer rimangano invariati);
- Permette una buona testabilità, ogni *layer* può essere isolato e testato separatamente;
- Il progetto non richiede un'elevata *scalabilità*$_G$ del sistema.

I *layer* implementati all'interno del sistema sono i seguenti:

- *API layer*: rappresenta la logica che riceve le richieste HTTP e orchestra le chiamate ai servizi disponibili;
- *Business layer*: rappresenta la logica di business dell'applicazione, riceve i dati dall'*API layer* ed effettua le operazioni necessarie;
- *Persistence layer*: rappresenta la connessione tra il *business layer* e il *database*, si occupa di eseguire le *query* e orchestrare l'utilizzo del *Object-Relational Mapper*;

- *Database layer*: rappresenta il *database* dell'applicativo, dove i dati sono salvati e prelevati.

Ogni *layer* interagisce solo con i *layer* sottostanti, pertanto è stato adottato un approccio di tipo *closed layer*.

5.3.2 Inversione del controllo

È stato fatto utilizzo del sistema di *dependency injection*$_G$ messo a disposizione dal *framework* FastAPI. In particolare il file jwt.py espone le rotte `/verifyToken` e `/token` e delega le operazioni di verifica e generazione dei *JSON Web Token* alle funzioni fornite dalla classe `JWTDependencies`. Tramite l'oggetto `Depends` il router richiede le dipendenze `get_verified_payload` per verificare il *token* e `get_new_token` per generarne uno nuovo, lasciando che FastAPI risolva e inietti gli oggetti necessari prima dell'esecuzione dell'endpoint. Questo approccio sposta all'esterno la creazione delle dipendenze, permettendo di sostituire facilmente l'implementazione durante i test o in fase di configurazione dell'applicazione.

```
1  #Definizione di un router API
2  jwt = APIRouter()
3  #Rotta API per verificare il token
4  @jwt.post("/verifyToken")
5  async def check_token(payload: dict =
   ↪  Depends(jwt_dependencies.get_verified_payload)):
6      return {"message": "Token verified", "payload": payload}
7  #Rotta API per ottenere un nuovo token
8  @jwt.post("/token")
9  async def get_token(payload: dict =
   ↪  Depends(jwt_dependencies.get_new_token)):
10     return payload
```

Codice 5.1: File jwt.py

```python
class JWTDependencies:
    #Funzione che controlla il token
    def get_verified_payload(
        self,
        credentials: HTTPAuthorizationCredentials =
        ↪ Security(HTTPBearer()),
    ) -> Dict[str, Any]:
        service = JWTService()
        success, result =
        ↪ service.verify_token(credentials=credentials)
        if not success:
            raise HTTPException(status_code=401,
            ↪ detail=result["message"])
        return result

    #Funzione per il rilascio di un nuovo token
    def get_new_token(
        self,
        client_id: str = Form(...),
        client_secret: str = Form(...),
        session: Session = Depends(db_dependencies.get_session),
    ) -> Dict[str, Any]:
        service = JWTService(session=session)
        success, result = service.generate_token(
            client_id=client_id, client_secret=client_secret
        )
        if not success:
            raise HTTPException(status_code=401,
            ↪ detail=result["message"])
        return result
```

Codice 5.2: Classe JWTDependencies

5.3.3 Design pattern comportamentali

È stato usato lo *strategy pattern* per definire un'interfaccia per l'algoritmo di ordinamento. Tale scelta permetterà in futuro, se necessario, di riscrivere completamente la logica di schedulazione mantenendo la compatibilità con la restante logica del modulo, a patto che venga implementata la stessa interfaccia. La definizione viene fornita nel file `ordering_alg.py`.

```python
class OrderingAlg(ABC):
    @abstractmethod
    def schedule_jobs(
        self,
        jobs: List[Job],
        printers: List[Printer],
        base_datetime: datetime,
        initial_schedule: Optional[SchedulingResult] = None,
        is_last_schedule: bool = False,
        future_capacity_by_order: Optional[Dict[int, int]] = None,
    ) -> Tuple[bool, SchedulingResult]:
        pass
```

Codice 5.3: Interfaccia schedule_jobs

La lista dei parametri accettati è la seguente:

- `jobs`: *array* di lavori da pianificare;
- `printers`: *array* di stampanti disponibili;
- `base_datetime`: data e ora di riferimento da cui far partire la schedulazione;
- `initial_schedule`: eventuale risultato già calcolato da integrare;
- `is_last_schedule`: valore booleano che indica se l'elaborazione corrente è l'ultima;

- `future_capacity_by_order`: dizionario opzionale con chiave l'identificativo dell'ordine e valore la capacità già riservata in pianificazioni future.

5.3.4 Design pattern strutturali

È stato usato il *repository pattern*, tale approccio consiste nel creare delle classi (*repository*) dotate dei metodi necessari per leggere o scrivere il *database*. L'adozione di questo *design pattern* porta a diversi vantaggi, tra cui:

- Semplifica la logica di accesso ai dati, esponendo un'interfaccia consistente per la lettura e la scrittura;
- Migliora la testabilità del sistema, in quanto ogni *repository* può essere sostituito facilmente da un *mock*;
- Migliora la pulizia e la leggibilità del codice;
- Permette di ridurre il tempo necessario ad un eventuale cambio di *database*, in quanto necessiterebbe solo di una riscrittura delle classi *repository*.

5.4 Struttura delle cartelle

La struttura delle cartelle adottata per il progetto è la seguente:

- `Mugalab-Automation`: cartella principale, contiene i file relativi alle *variabili d'ambiente*, alla configurazione di Docker, il file `requirements.txt`, utilizzato per tenere traccia di tutte le dipendenze del sistema e il file `.gitignore` che riferisce al *version control system* quali elementi ignorare;
  - `app`: cartella contenente tutto il codice sorgente;
    - `api`: cartella contenente i file che dichiarano le rotte *API*;
    - `contexts`: cartella contenente i file che dichiarano gli oggetti utilizzati come contesto;
    - `db`: cartella che contiene i file che dichiarano i dati per la connessione con il *database*;
    - `dependencies`: cartella che contiene i file che dichiarano le classi e le funzioni utilizzate per effettuare *dependency injection*;
    - `interfaces`: cartella che contiene i file che dichiarano le interfacce;
    - `models`: cartella che contiene i file che dichiarano i *modelli*, ovvero le classi che rappresentano le entità di dominio, validano i dati in ingresso e definiscono lo schema persistito nel database;
    - `repositories`: cartella che contiene i file che dichiarano i *repository* usati per leggere e/o scrivere dati nel *database*;
    - `services`: cartella che contiene le classi che contengono tutta la *logica di business*$_G$ dell'applicazione;
    - `tests`: cartella che contiene i *test funzionali* prodotti.

La struttura delle cartelle del progetto è illustrata nella figura 5.1.

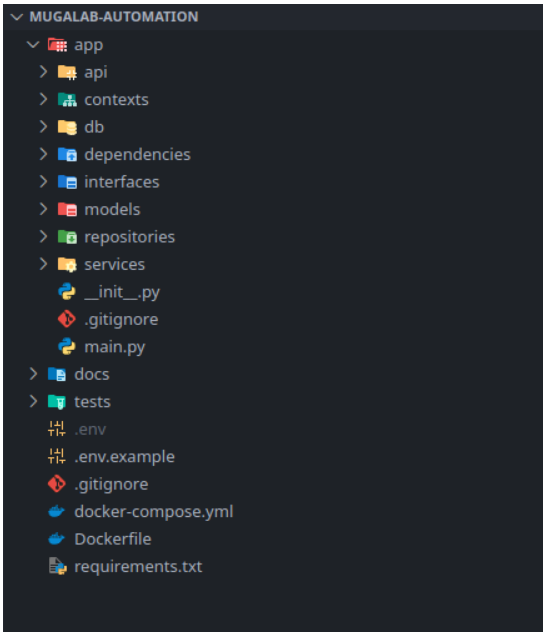


Figura 5.1: Struttura delle cartelle del progetto

5.5 Lista dei modelli

In questa sezione vengono descritti tutti i *modelli*.

5.5.1 Client

5.5.1.1 Scopo del modello

Rappresenta un *client* registrato e autorizzato ad utilizzare le rotte *API* esposte dal sistema. La struttura di questo *modello* definisce la tabella `clients` (vedi sezione 5.2.1).

5.5.1.2 Definizione

```python
class Client(SQLModel, table=True):
    __tablename__ = "clients"
    id: int | None = Field(default=None, primary_key=True)
    client_id: str = Field(nullable=False, unique=True, index=True)
    client_secret: str = Field(nullable=False)
    created_at: datetime = Field(
        default_factory=datetime.now,
        sa_column=Column(DateTime, server_default=func.now()),
    )
    is_active: bool = Field(default=True, sa_column=Column(Boolean,
    ↪  server_default="1"))
```

Codice 5.4: Definizione del model Client

5.5.2 JWTConfig

5.5.2.1 Scopo del modello

Questo *modello* definisce ed espone la configurazione dei *JSON Web Token*, in particolare definisce la chiave e l'algoritmo utilizzati per la firma e la scadenza in minuti.

5.5.2.2 Definizione

```python
@dataclass
class JWTConfig:
    secret_key: str = os.getenv("JWT_SECRET_KEY", "default-secret")
    algorithm: str = "HS256"
    expire_minutes: int = 60
```

Codice 5.5: Definizione del model JWTConfig

5.5.3 ScheduleRequest

5.5.3.1 Scopo del modello

Questo *modello* definisce e valida la struttura del corpo della richiesta da inviare per richiedere una pianificazione. In particolare ogni oggetto di tipo `ScheduleRequest` deve contenere:

- Un *array* di `Order` (vedi 5.5.9);
- Un *array* di `Printer` (vedi 5.5.13);
- Un intero che specifica l'identificativo dell'utente che effettua la richiesta;
- Una stringa che specifica il *token* da utilizzare per l'invio dei dati al gestionale.

5.5.3.2 Definizione

```
1 class ScheduleRequest(BaseModel):
2     orders: List[Order]
3     printers: List[Printer]
4     user_id: int
5     callback_token: str
```

Codice 5.6: Definizione del modello ScheduleRequest

5.5.4 ScheduleResponse

5.5.4.1 Scopo del modello

Questo *modello* definisce e valida la struttura del corpo della risposta, contenente la pianificazione, inviata al gestionale una volta terminato il calcolo dei risultati. In particolare ogni oggetto di tipo `ScheduleResponse` deve contenere:

- Un risultato di tipo `SchedulingResult` (vedi 5.5.14)
- Un *array* di `Order` (vedi 5.5.9);

- Un intero che specifica l'identificativo dell'utente che deve ricevere la pianificazione.

5.5.4.2 Definizione

```
class ScheduleResponse(BaseModel):
    result: SchedulingResult
    orders: List[Order]
    user_id: int

```

Codice 5.7: Definizione del modello ScheduleResponse

5.5.5 PrinterSchedule

5.5.5.1 Scopo del modello

Questo *modello* definisce e valida la schedulazione proposta per una singola stampante. In particolare ogni oggetto di tipo `PrinterSchedule` deve contenere:

- Un *array* di `ScheduledJob` (vedi 5.5.6);
- Un intero che indica i minuti di ritardo accumulati su quella specifica stampante;
- Un intero che specifica il conteggio dei lavori pianificati su quella stampante.

5.5.5.2 Definizione

```
class PrinterSchedule(BaseModel):
    jobs: List[ScheduledJob]
    total_delay_minutes: int
    job_count: int
```

Codice 5.8: Definizione del modello PrinterSchedule

5.5.6 ScheduledJob

5.5.6.1 Scopo del modello

Questo *modello* definisce e valida una singola istanza di un lavoro pianificato. In particolare ogni oggetto di tipo `ScheduledJob` deve contenere:

- Una stringa che indica il nome del lavoro;
- Una stringa che indica il materiale;
- Un oggetto di tipo `datetime` che indica data e ora di inizio del lavoro;
- Un oggetto di tipo `datetime` che indica data e ora di fine del lavoro;
- Un intero che indica la durata della stampa in minuti;
- Un intero che indica il ritardo nella esecuzione del lavoro in minuti;
- Un oggetto di tipo `Order` (vedi 5.5.9);
- Un oggetto di tipo `datetime` che indica la scadenza del lavoro;
- Un oggetto di tipo `Plate` (vedi 5.5.10).

5.5.6.2 Definizione

```
1  class ScheduledJob(BaseModel):
2      job_name: str
3      material: str
4      start_time: datetime
5      end_time: datetime
6      duration_minutes: int
7      delay_minutes: int
8      order: Order
9      deadline: datetime
10     plate: Plate
```

Codice 5.9: Definizione del modello ScheduledJob

5.5.7 Summary

5.5.7.1 Scopo del modello

Questo *modello* definisce e valida le statistiche finali della pianificazione richiesta. In particolare ogni oggetto di tipo `Summary` deve contenere:

- Un intero che indica il numero totale di minuti di ritardo;
- Un intero che indica il numero totale di lavori pianificati;
- Un intero che indica lo stato del solutore (in base al calcolo di una soluzione ottima, accettabile o non accettabile);
- Un intero che indica il numero totale di cambi di materiale;
- Un intero che indica il $makespan_G$ in minuti;

5.5.7.2 Definizione

```python
class Summary(BaseModel):
    total_delay_minutes: int
    total_jobs: int
    solver_status: int
    total_material_changes: int
    total_makespan_minutes: int
```

Codice 5.10: Definizione del modello Summary

5.5.8 Job

5.5.8.1 Scopo del modello

Questo *modello* definisce e valida una singola istanza di un lavoro da pianificare. In particolare ogni oggetto di tipo `Job` deve contenere:

- Una stringa che indica il nome del lavoro;

- Un oggetto di tipo `time` che indica durata della stampa;
- Un oggetto di tipo `Plate` (vedi 5.5.10);
- Un oggetto di tipo `datetime` che indica la scadenza del lavoro;
- Un oggetto di tipo `Order` (vedi 5.5.9).

5.5.8.2 Definizione

```
1 class Job(BaseModel):
2     name: str
3     time: time
4     material: str
5     plate: Plate
6     end_time: datetime
7     order: Order
```

Codice 5.11: Definizione del modello Job

5.5.9 Order

5.5.9.1 Scopo del modello

Questo `modello` definisce e valida una singola istanza di un ordine di produzione. In particolare ogni oggetto di tipo `Order` deve contenere:

- Un intero che indica l'identificativo dell'ordine;
- Un oggetto di tipo `date` che indica la scadenza dell'ordine;
- Una stringa che indica il codice dell'ordine;
- Un intero che indica i pezzi ordinati;
- Un intero che indica i pezzi già stampati;
- Un oggetto di tipo `PrinterWork` (vedi 5.5.12).

5.5.9.2 Definizione

```
class Order(BaseModel):
    id: int
    expiration_date: date
    number: str
    pcs_ordered: int
    pcs_printed: int
    printer_work: PrinterWork
```

Codice 5.12: Definizione del modello Order

5.5.10 Plate

5.5.10.1 Scopo del modello

Questo *modello* definisce e valida una singola istanza di un piatto di stampa. In particolare ogni oggetto di tipo `Plate` deve contenere:

- Un intero che indica l'identificativo del piatto;
- Un intero che indica il numero di pezzi presenti nel piatto;
- Un oggetto di tipo `time` che indica il tempo necessario per stampare un piatto.

5.5.10.2 Definizione

```
class Plate(BaseModel):
    id: int
    pcs: int
    time: time
```

Codice 5.13: Definizione del modello Plate

5.5.11 PrinterMaterial

5.5.11.1 Scopo del modello

Questo *modello* valida e definisce una singola istanza di un materiale utilizzato per la stampa. In particolare ogni oggetto di tipo `PrinterMaterial` deve contenere:

- Un intero che indica l'identificativo del materiale;
- Una stringa che indica il nome del materiale.

5.5.11.2 Definizione

```
1 class PrinterMaterial(BaseModel):
2     id: int
3     name: str
```

Codice 5.14: Definizione del modello PrinterMaterial

5.5.12 PrinterWork

5.5.12.1 Scopo del modello

Questo *modello* valida e definisce una singola istanza di un articolo da stampare. In particolare ogni oggetto di tipo `PrinterWork` deve contenere:

- Un intero che indica l'identificativo dell'articolo;
- Una stringa che indica il nome dell'articolo;
- Un oggetto di tipo `PrinterMaterial` (vedi 5.5.11);
- Un *array* di oggetti di tipo `Plate` (vedi 5.5.10) che indica i formati di piatti di stampa disponibili per l'articolo.

5.5.12.2 Definizione

```python
class PrinterWork(BaseModel):
    id: int
    work_name: str
    printer_material: PrinterMaterial
    plates: List[Plate]
```

Codice 5.15: Definizione del modello PrinterWork

5.5.13 Printer

5.5.13.1 Scopo del modello

Questo *modello* definisce e valida una singola istanza di una stampante. In particolare ogni oggetto di tipo `Printer` deve contenere:

- Un intero che indica l'identificativo della stampante;
- Una stringa che indica il nome della stampante.

5.5.13.2 Definizione

```python
class Printer(BaseModel):
    id: int
    name: str
```

Codice 5.16: Definizione del modello Printer

5.5.14 SchedulingResult

5.5.14.1 Scopo del modello

Questo *modello* valida e definisce una risultato di schedulazione. In particolare ogni oggetto di tipo `SchedulingResult` deve contenere:

- Un oggetto di tipo `Summary` (vedi 5.5.7);

- Un dizionario che associa ad ogni stampante un oggetto di tipo `PrinterSchedule` (vedi 5.5.5)

5.5.14.2 Definizione

```
1 class SchedulingResult(BaseModel):
2     summary: Summary
3     printers: Dict[str, PrinterSchedule]
```

Codice 5.17: Definizione del modello SchedulingResult

5.5.15 TaskVariable

5.5.15.1 Scopo del modello

Questo *modello* definisce una variabile utilizzata dalla libreria Google OR-Tools (vedi 5.1.2.6) per identificare un lavoro da pianificare. In particolare ogni oggetto di tipo `TaskVariable` deve contenere:

- Un oggetto di tipo `IntVar` che rappresenta l'istante di inizio del lavoro espresso in minuti rispetto all'orario di inizio della schedulazione;
- Un oggetto di tipo `IntVar` che rappresenta l'istante di fine del lavoro espresso nello stesso riferimento temporale;
- Un oggetto di tipo `IntVar` che agisce come variabile booleana per indicare se il lavoro è stato assegnato a una stampante;
- Un oggetto di tipo `IntervalVar` che collega inizio, durata e fine della lavorazione e permette al solutore di imporre i vincoli di non sovrapposizione;
- Una stringa che indica il materiale richiesto dal lavoro;
- Un intero che indica la durata del lavoro in minuti;
- Un intero che rappresenta la deadline del lavoro in minuti rispetto all'orario di inizio della schedulazione;

- Un oggetto di tipo `Job` (vedi 5.5.8) che mantiene un riferimento al lavoro originale.

5.5.15.2 Definizione

```python
@dataclass
class TaskVariable:
    start: IntVar
    end: IntVar
    assigned: IntVar
    interval: IntervalVar
    material: str
    duration: int
    deadline: int
    job_instance: Job
```

Codice 5.18: Definizione del modello TaskVariable

5.5.16 SchedulerConfig

5.5.16.1 Scopo del modello

Questo *modello* definisce la configurazione delle opzioni dello schedulatore. In particolare ogni oggetto di tipo `SchedulerConfig` deve contenere:

- Un intero che indica quanti minuti possiede una giornata;
- Un intero che indica dopo quanti minuti dall'inizio della giornata comincia il turno lavorativo;
- Un intero che indica dopo quanti minuti dall'inizio della giornata finisce il turno lavorativo;
- Un intero che indica la penalità per un cambio di materiale;
- Un intero che indica la penalità per ogni minuto di ritardo accumulato;

- Un intero che indica quanti minuti servono per un cambio piatto;
- Un numero a virgola mobile che indica quanti secondi può impiegare il solutore per la ricerca della soluzione ottima;
- Un valore booleano che indica se il solutore deve stampare a terminale il progresso della ricerca.

5.5.16.2 Definizione

```python
@dataclass
class SchedulerConfig:
    """
    Classe che gestisce la configurazione dello scheduler
    """

    minutes_per_day: int = int(os.getenv("MINUTES_PER_DAY", 24 * 60))
    jobs_start_time: int = int(os.getenv("JOBS_START_TIME", 9 * 60))
    jobs_end_time: int = int(os.getenv("JOBS_END_TIME", 18 * 60))
    material_change_penalty: int =
    ↪  int(os.getenv("MATERIAL_CHANGE_PENALTY", 100))
    late_penalty_per_minute: int =
    ↪  int(os.getenv("LATE_PENALTY_PER_MINUTE", 1))
    change_plate_time: int = int(os.getenv("CHANGE_PLATE_TIME", 5))
    solver_timeout: float = float(os.getenv("SOLVER_TIMEOUT", 300.0))
    log_search: bool = False
```

Codice 5.19: Definizione del modello TaskVariable

5.6 Servizi

In questa sezione vengono descritti i *servizi* più rilevanti.

5.6.1 OrderConversionService

5.6.1.1 Scopo

Questo *servizio* ha lo scopo di incapsulare la logica di conversione di un *array* di oggetti di tipo `Order` ad un *array* di oggetti di tipo `Job`, operazione necessaria per ottenere tutti i possibili lavori da pianificare nella coda di stampa. Il metodo `convert_orders_to_jobs` scansiona tutti gli ordini ricevuti come parametro, per ogni ordine controlla le tipologie di piatti stampabili per l'articolo richiesto e calcola quanti piatti sono necessari per completare la richiesta per ogni variante di piatto. Vengono poi costruiti gli oggetti di tipo `Job`, i quali potranno poi essere inseriti o meno nella coda di stampa in base alla richiesta.

5.6.1.2 Implementazione

```
1  class OrderConversionService:
2      def convert_orders_to_jobs(self, orders: List[Order]) ->
       ↪  List[Job]:
3          result: List[Job] = []
4          for order in orders:
5              for plate in order.printer_work.plates:
6                  plate_count = order.get_plate_count(plate=plate)
7                  for i in range(plate_count):
8                      result.append(
9                          Job(
10                             name=f"Job_{i}_Order_{order.number}",
11                             time=plate.time,
12                             end_time=datetime.combine(
13                                 date=(order.expiration_date -
                                   ↪  timedelta(1)),
14                                 time=time(23, 59, 59),
15                             ),
16                             plate=plate,
17                             material=order.printer_work.printer_mater
                               ↪  ial.name,
```

```
                            order=order,
                        )
                    )
        return result
```

Codice 5.20: Implementazione del servizio OrderConversionService

5.6.2 SchedulingService

5.6.2.1 Scopo

Questo *servizio* ha lo scopo di orchestrare il processo di schedulazione. Il metodo `schedule_with_batches` accetta come parametri un *array* di `Job`, un *array* di `Printer` e un oggetto di tipo `OrderingAlg` per la scelta dell'algoritmo da utilizzare (*strategy pattern*). I lavori da pianificare vengono divisi in insiemi (detti *batch*) di dimensione fissa, questa scelta implementativa permette di ottenere una soluzione qualitativamente inferiore a fronte di un forte incremento delle prestazioni, la dimensione dei batch rimane comunque configurabile attraverso la gestione delle *variabili d'ambiente*. Ogni *batch* viene poi pianificato separatamente e unito agli altri. Gli altri metodi presenti nella classe (qui omessi per semplicità) sono:

- `__get_future_capacity_by_order`: si occupa di calcolare la produzione possibile dei *batch* futuri, per evitare sovra-produzione;
- `__merge_schedules`: si occupa di unire due schedulazioni appartenenti a *batch* differenti;
- `__get_total_material_changes`: calcola il numero totale di cambi di materiale;
- `__get_total_makespan`: calcola il *makespan* totale;
- `__split_in_batches`: divide l'*array* di `Job` in *batch*;
- `__set_summary`: imposta l'oggetto `Summary` della soluzione, inserendo delle informazioni riassuntive sulla pianificazione.

5.6.2.2 Implementazione

```python
class SchedulingService:
    def schedule_with_batches(
        self,
        jobs: List[Job],
        printers: List[Printer],
        alg: OrderingAlg,
    ) -> Tuple[bool, SchedulingResult]:
        batches = self.__split_in_batches(jobs=jobs)

        final_result: SchedulingResult | None = None
        last_schedule_state: SchedulingResult | None = None
        base_datetime = datetime.now().replace(
            hour=0, minute=0, second=0, microsecond=0
        )

        suffix_sums_per_batch =
        ↪  self.__get_future_capacity_by_order(batches=batches)

        for i, batch in enumerate(batches): #Divisione in batch
            future_capacity_by_order = suffix_sums_per_batch[i]
            is_last = i == len(batches) - 1
            success, new_schedule = alg.schedule_jobs(
                jobs=batch,
                printers=printers,
                initial_schedule=last_schedule_state,
                base_datetime=base_datetime,
                is_last_schedule=is_last,
                future_capacity_by_order=future_capacity_by_order,
            )
            if not success:
                raise Exception(f"Errore durante la schedulazione del
                ↪  lotto {i + 1}")
```

```
31
32              if not final_result:
33                  final_result = new_schedule
34              else:
35                  final_result = self.__merge_schedules(final_result,
                    ↪  new_schedule)
36              last_schedule_state = final_result
37
38          if not final_result:
39              raise Exception("Result is not a SchedulingResult")
40
41          final_result = self.__set_summary(final_result, base_datetime)
            ↪  #Calcolo del Summary
42          return True, final_result
```

Codice 5.21: Implementazione del servizio SchedulingService

5.6.3 ORToolsPrintingScheduler

5.6.3.1 Scopo

Questo *servizio* ha lo scopo di utilizzare la libreria Google OR-Tools per istanziare le variabili necessarie per il modello CP-SAT, impostare i vincoli necessari, definire la funzione obiettivo e chiamare il solutore. Il servizio estende l'interfaccia `OrderingAlg`, pertanto definisce obbligatoriamente un metodo `schedule_jobs` con i parametri descritti nella sezione 5.3. Tale metodo si occupa di calcolare l'orizzonte temporale della soluzione (tramite il metodo `__calculate_horizon`) e di istanziare i servizi necessari (`VariableManager`, `ConstraintManager`, `SolutionFormatter` e `ObjectiveManager`) e orchestare le chiamate ai loro metodi pubblici.

5.6.3.2 Implementazione

```
class ORToolsPrintingScheduler(OrderingAlg):
    def __init__(self):
        self.config = SchedulerConfig()

    def schedule_jobs(
        self,
        jobs: List[Job],
        printers: List[Printer],
        base_datetime: datetime,
        initial_schedule: Optional[SchedulingResult] = None,
        is_last_schedule: bool = False,
        future_capacity_by_order: Optional[Dict[int, int]] = None,
    ) -> Tuple[bool, SchedulingResult]:
        horizon = self.__calculate_horizon(
            jobs=jobs, printers=printers, base_datetime=base_datetime
        )

        context = SchedulerContext.create(
```

```
            config=self.config, horizon=horizon,
            ↪ base_datetime=base_datetime
        )

        var_manager = VariableManager(context=context)
        tasks = var_manager.create_variables(jobs=jobs,
        ↪ printers=printers)

        constraint_manager = ConstraintManager(context=context)
        if initial_schedule:
            constraint_manager.add_initial_state_constraints(
                tasks=tasks,
                printers=printers,
                initial_schedule=initial_schedule,
            )
        constraint_manager.add_basic_constraints(
            tasks=tasks, jobs=jobs, printers=printers
        )
        constraint_manager.add_production_constraints(
            tasks=tasks,
            jobs=jobs,
            printers=printers,
            last_schedule_state=initial_schedule,
            future_capacity_by_order=future_capacity_by_order,
        )

        obj_manager = ObjectiveManager(context=context)
        obj_manager.define_objective(
            tasks=tasks,
            jobs=jobs,
            printers=printers,
            final_batch=is_last_schedule,
            last_schedule_state=initial_schedule,
            future_capacity_by_order=future_capacity_by_order,
```

```
51          )
52
53          if self.config.log_search:
54              context.solver.parameters.log_search_progress = True
55          context.solver.parameters.max_time_in_seconds = float(
56              self.config.solver_timeout
57          )
58
59          status = context.solver.Solve(context.model)
60
61          formatter = SolutionFormatter(context=context)
62          result = formatter.get_solution_dict(context.solver, tasks,
            ↪  printers, status)
63          success = status in (cp_model.OPTIMAL, cp_model.FEASIBLE)
64          return success, result
```

Codice 5.22: Implementazione del servizio ORToolsPrintingScheduler

5.6.4 ConstraintManager

5.6.4.1 Scopo

Questo *servizio* ha lo scopo di dichiarare i vincoli della soluzione da trovare. Il metodo `add_initial_state_constraints` accetta come parametri un dizionario che associa la tupla nome lavoro-nome stampante al corrispondente oggetto di tipo `TaskVariable`, un *array* di `Printer` e un oggetto di tipo `SchedulingResult`. Il suo scopo è quello di assicurarsi mediante la definizione degli appositi vincoli che ogni lavoro sia pianificato dopo il suo predecessore e che tra due lavori consecutivi passi il tempo necessario per il cambio piatto. Il metodo `add_basic_constraints` accetta come parametri un dizionario che associa la tupla nome lavoro-nome stampante al corrispondente oggetto di tipo `TaskVariable`, un *array* di `Printer` e un *array* di `Job`. Il suo scopo è quello di orchestrare le chiamate ai metodi privati `__add_assignment_constraints`, `__add_no_overlap_constraints`,

`__add_start_time_constraints` e `__create_precedence_constraints`, quest'ultimi si occupano, rispettivamente, di definire i vincoli che impongono che ogni lavoro sia pianificato su una sola stampante, non si sovrapponga con altri e che inizi durante la giornata lavorativa.

Il metodo `add_production_constraints` accetta come parametri un dizionario che associa la tupla nome lavoro-nome stampante al corrispondente oggetto di tipo `TaskVariable`, un *array* di `Printer`, un oggetto opzionale di tipo `SchedulingResult` e un dizionario che indica, per l'ordine specificato, quanti pezzi sono già previsti nei *batch* successivi. Il suo scopo è quello di assicurarsi, mediante la definizione dei vincoli necessari, che la produzione copra la richiesta anche in caso di divisione in *batch*.

5.6.4.2 Implementazione

```
class ConstraintManager:
    def __init__(self, context: SchedulerContext):
        self.context = context

    def add_initial_state_constraints(
        self,
        tasks: Dict[Tuple[str, str], TaskVariable],
        printers: List[Printer],
        initial_schedule: SchedulingResult,
    ):
        plate_change_time = self.context.config.change_plate_time
        unique_job_names = {key[0] for key in tasks.keys()}
        for printer in printers:
            if (
                printer.name in initial_schedule.printers.keys()
                and initial_schedule.printers[printer.name].jobs
            ):
                printer_schedule =
                ↪  initial_schedule.printers[printer.name]
```

```
                last_end_time = max(job.end_time for job in
                ↪ printer_schedule.jobs)
                release_time_minutes = max(
                    0,
                    int(
                        (last_end_time - self.context.base_datetime).
                        ↪ total_seconds()
                        // 60
                    ),
                )
                for job_name in unique_job_names:
                    task_key = (job_name, printer.name)
                    if task_key in tasks:
                        task_start_var = tasks[task_key].start
                        self.context.model.Add(
                            task_start_var >= release_time_minutes +
                            ↪ plate_change_time
                        ).OnlyEnforceIf(tasks[task_key].assigned)

    def add_basic_constraints(
        self,
        tasks: Dict[Tuple[str, str], TaskVariable],
        jobs: List[Job],
        printers: List[Printer],
    ):
        jobs_map: Dict[str, Job] = {j.name: j for j in jobs}
        self.__add_assignment_constraints(tasks, jobs_map, printers)
        self.__add_no_overlap_constraints(tasks, jobs_map, printers)
        self.__add_start_time_constraints(tasks, jobs_map, printers)

    def add_production_constraints(
        self,
        tasks: Dict[Tuple[str, str], TaskVariable],
        jobs: List[Job],
```

```
50          printers: List[Printer],
51          last_schedule_state: Optional[SchedulingResult],
52          future_capacity_by_order: Optional[Dict[int, int]],
53      ):
54          job_map: Dict[str, Job] = {j.name: j for j in jobs}
55          orders: Dict[int, Set[str]] = {}
56          for job in jobs:
57              oid = job.order.id
58              orders.setdefault(oid, set()).add(job.name)
59          produced_so_far: Dict[int, int] = {}
60          if last_schedule_state:
61              for printer_data in last_schedule_state.printers.values():
62                  for job in printer_data.jobs:
63                      oid = job.order.id
64                      pcs = job.plate.pcs
65                      produced_so_far[oid] = produced_so_far.get(oid, 0)
                        ↪  + pcs
66          future_cap = future_capacity_by_order or {}
67          for oid, job_names in orders.items():
68              any_job = next(iter(job_names))
69              order = job_map[any_job].order
70              already_done = produced_so_far.get(order.id, 0)
71              residual = max(0, order.get_remaining_pcs() -
                ↪  already_done)
72              cap_future = max(0, int(future_cap.get(order.id, 0)))
73              need_now = max(0, residual - cap_future)
74              if need_now == 0:
75                  continue
76              produced_terms = []
77              for j in job_names:
78                  pcs = job_map[j].plate.pcs
79                  for p in printers:
80                      produced_terms.append(tasks[(j, p.name)].assigned
                        ↪  * int(pcs))
```

```
81              self.context.model.Add(sum(produced_terms) >= need_now)
```

Codice 5.23: Implementazione del servizio ConstraintManager

5.6.5 VariableManager

5.6.5.1 Scopo

Questo *servizio* ha lo scopo di convertire gli oggetti di tipo `Job` a degli oggetti di tipo `TaskVariable`. Il metodo `create_variables` accetta come parametri un *array* di `Job` e un *array* di `Printer`. Per ogni coppia `Job-Printer` viene chiamato il metodo `__create_task_variables` il quale si occupa dell'effettiva creazione dell'oggetto di tipo `TaskVariable`.

5.6.5.2 Implementazione

```
1  class VariableManager:
2      """Gestisce la creazione delle variabili del modello"""
3
4      def __init__(self, context: SchedulerContext):
5          self.context = context
6
7      def create_variables(
8          self, jobs: List[Job], printers: List[Printer]
9      ) -> Dict[Tuple[str, str], TaskVariable]:
10         """Crea tutte le variabili necessarie per il modello"""
11         tasks: Dict[Tuple[str, str], TaskVariable] = {}
12
13         for job in jobs:
14             for printer in printers:
15                 task_vars = self.__create_task_variables(job, printer)
16                 tasks[(job.name, printer.name)] = task_vars
17
18         return tasks
19
```

```
20      def __create_task_variables(self, job: Job, printer: Printer) ->
        ↪  TaskVariable:
21          """Crea le variabili per un singolo task"""
22          job_hash = hash(job.name + printer.name) % 100000
23
24          start = self.context.model.NewIntVar(0, self.context.horizon,
            ↪  f"s_{job_hash}")
25          end = self.context.model.NewIntVar(0, self.context.horizon,
            ↪  f"e_{job_hash}")
26          assigned = self.context.model.NewBoolVar(f"a_{job_hash}")
27          interval = self.context.model.NewOptionalIntervalVar(
28              start,
29              job.get_time_minutes(),
30              end,
31              assigned,
32              f"i_{job_hash}",
33          )
34
35          return TaskVariable(
36              start=start,
37              end=end,
38              assigned=assigned,
39              interval=interval,
40              material=job.material,
41              duration=job.get_time_minutes(),
42              deadline=job.get_deadline_minutes(self.context.base_datet
                ↪  ime),
43              job_instance=job,
44          )
```

Codice 5.24: Implementazione del servizio VariableManager

5.6.6 ObjectiveManager

5.6.6.1 Scopo

Questo *servizio* si occupa di gestire la definizione della *funzione obiettivo*$_G$. Il metodo `define_objective` accetta come parametri un dizionario che associa la tupla nome lavoro-nome stampante al corrispondente oggetto di tipo `TaskVariable`, un *array* di `Printer`, un *array* di `Job`, una variabile booleana che indica se il *batch* esaminato è l'ultimo, l'ultimo *batch* pianificato, un dizionario che indica, per l'ordine specificato, quanti pezzi sono già previsti nei *batch* successivi e l'ultimo `SchedulingResult` calcolato. Tale metodo si occupa quindi di orchestrare le chiamate ai metodi privati `__add_timing_penalties`, `__add_makespan_objective` e `__add_production_objective` i quali si occupano rispettivamente di definire le penalità per i ritardi, aggiungere l'obiettivo di minimizzare il *makespan* e minimizzare la sovraproduzione.

5.6.6.2 Implementazione

```
class ObjectiveManager:
    def __init__(self, context: SchedulerContext):
        self.context = context

    def define_objective(
        self,
        tasks: Dict[Tuple[str, str], TaskVariable],
        jobs: List[Job],
        printers: List[Printer],
        final_batch: bool,
        last_schedule_state: Optional[SchedulingResult] = None,
        future_capacity_by_order: Optional[Dict[int, int]] = None,
    ) -> None:
        obj_terms: List = []
        jobs_map: Dict[str, Job] = {j.name: j for j in jobs}
        self.__add_timing_penalties(
```

```
17              tasks=tasks, jobs_data=jobs_map, printers=printers,
                ↪  obj_terms=obj_terms
18          )
19          self.__add_makespan_objective(
20              tasks=tasks, jobs_data=jobs_map, printers=printers,
                ↪  obj_terms=obj_terms
21          )
22          self.__add_production_objective(
23              tasks=tasks,
24              jobs_data=jobs_map,
25              printers=printers,
26              obj_terms=obj_terms,
27              final_batch=final_batch,
28              last_schedule_state=last_schedule_state,
29              future_capacity_by_order=future_capacity_by_order,
30          )
31          self.context.model.Minimize(sum(obj_terms))
```

Codice 5.25: Implementazione del servizio ObjectiveManager

5.6.7 SolutionFormatter

5.6.7.1 Scopo

Questo *servizio* si occupa di gestire la formattazione della soluzione e la creazione dell'oggetto `SchedulingResult`. Il metodo `get_solution_dict` accetta come parametri l'istanza del solutore, un dizionario che associa la tupla nome lavoro-nome stampante al corrispondente oggetto di tipo `TaskVariable`, un *array* di `Printer` e una variabile `status` che indica lo stato del solutore. Tale metodo si occupa quindi della costruzione della soluzione orchestrando le chiamate ai metodi `__get_status_string`, `__get_printer_schedule` e `__update_job_start_times` i quali si occupano, rispettivamente, di convertire lo stato del solutore in una stringa, di costruire l'oggetto `PrinterSchedule` e aggiornare gli oggetti `Job` originali con gli orari di avvio calcolati.

5.6.7.2 Implementazione

```
class SolutionFormatter:
    def __init__(self, context: SchedulerContext):
        self.context = context

    def get_solution_dict(
        self,
        solver: cp_model.CpSolver,
        tasks: Dict[Tuple[str, str], TaskVariable],
        printers: List[Printer],
        status,
    ) -> SchedulingResult:
        result = SchedulingResult(
            status=self.__get_status_string(status),
            success=status in (cp_model.OPTIMAL, cp_model.FEASIBLE),
            printers={},
            summary=Summary(
                total_delay_minutes=0,
                total_jobs=0,
                solver_status=status,
                total_material_changes=0,
                total_makespan_minutes=0,
            ),
        )

        if result.success:
            total_late = 0
            total_jobs = 0
            for pr in printers:
                printer_result = self.__get_printer_schedule(solver,
                ↪  tasks, pr.name)
                result.printers[pr.name] = printer_result
                total_late += printer_result.total_delay_minutes
```

```
                total_jobs += len(printer_result.jobs)
            result.summary.total_delay_minutes = total_late
            result.summary.total_jobs = total_jobs
            self.__update_job_start_times(solver, tasks)
        return result


```

Codice 5.26: Implementazione del servizio SolutionFormatter

5.7 Rotte API

In questa sezione vengono descritte le rotte *API* esposte dal sistema.

5.7.1 `/schedule`

5.7.1.1 Scopo

Questa rotta *API* accetta come parametro un oggetto di tipo `ScheduleRequest`. Se la validazione ha successo viene avviata una schedulazione.

5.7.1.2 Implementazione

```
scheduling = APIRouter(dependencies=[Depends(jwt_dependencies.get_ver
↪  ified_payload)])

async def schedule_job(request: ScheduleRequest):
    jobs = OrderConversionService().convert_orders_to_jobs(orders=req
    ↪  uest.orders)
    status, result = SchedulingService().schedule_with_batches(
        jobs=jobs,
        printers=request.printers,
        alg=ORToolsPrintingScheduler(),
    )
    if status and len(request.printers) > 0 and len(request.orders) >
    ↪  0:
```

```
        response = ScheduleResponse(
            result=result, user_id=request.user_id,
            ↪ orders=request.orders
        ).clean_empty_printers()
        print(response)
        async with httpx.AsyncClient() as client:
            await client.post(
                os.getenv(
                    "WEBHOOK_URL", "http://idrotech-manager.ddev.site
                    ↪ /api/webhook" #URL di default ambiente locale
                    ↪ di sviluppo
                ),
                json={
                    "message": "Job scheduled successfully",
                    "data": response.model_dump_json(),
                },
                timeout=60,
                headers={"Authorization": f"Bearer
                ↪ {request.callback_token}"},
            )
    else:
        raise Exception(
            "Il modello non ha trovato una soluzione ottimale o
            ↪ accettabile.",
        )


@scheduling.post("/schedule")
async def schedule(request: ScheduleRequest, background_tasks:
↪ BackgroundTasks):
    background_tasks.add_task(schedule_job, request)
    return {"message": "Scheduling request accepted"}
```

Codice 5.27: Implementazione della rotta `/schedule`

5.7.2 /token

5.7.2.1 Scopo

Questa rotta *API* riceve come parametro le credenziali di un *client* registrato e, se sono corrette, rilascia un nuovo *JSON Web Token*.

5.7.2.2 Implementazione

```
1 @jwt.post("/token")
2 async def get_token(payload: dict =
  ↪ Depends(jwt_dependencies.get_new_token)):
3     return payload
```

Codice 5.28: Implementazione della rotta `/token`

5.7.3 /verifyToken

5.7.3.1 Scopo

Questa rotta *API* riceve come parametro il *JSON Web Token* inviato da un *client* registrato e ne effettua la verifica.

5.7.3.2 Implementazione

```
1 @jwt.post("/verifyToken")
2 async def check_token(payload: dict =
  ↪ Depends(jwt_dependencies.get_verified_payload)):
3     return {"message": "Token verified", "payload": payload}
```

Codice 5.29: Implementazione della rotta `/verifyToken`

5.8 Sicurezza del sistema

In questa sezione vengono descritte tutte le scelte implementative riguardanti la sicurezza del sistema.

5.8.1 Validazione delle richieste

Il *framework* FastAPI mette a disposizione una validazione automatica delle richieste inviate alle rotte *API*, tale procedura permette di scartare tutti gli input non conformi.

5.8.2 JSON Web Token

Per validare l'autore della richiesta è stata implementata una logica simile allo standard OAuth 2.0 Client Credentials Flow[4]. Il funzionamento di tale logica è il seguente:

- Il *client* che deve utilizzare il sistema invia le sue credenziali al modulo di pianificazione inserendo nel corpo della richiesta i valori `client_id` e `client_secret`;
- Il modulo di pianificazione valida le credenziali e risponde con un *JSON Web Token* firmato con una chiave segreta tramite l'algoritmo `HS256`;
- Il *client* può inserire nell'intestazione della richiesta (detta anche *header*) il *JSON Web Token* ricevuto;
- Il modulo di pianificazione può validare il *JSON Web Token* verificando la firma utilizzando la chiave segreta.

Le differenze rispetto allo standard sono le seguenti:

- Non viene effettuata la rotazione delle chiavi;
- Il modulo di pianificazione svolge anche la funzione di *authorization server*;

È stato scelto assieme al tutor aziendale di non implementare lo standard nella sua totalità per riservare maggior tempo di sviluppo alle restanti funzionalità.

[4] *OAuth 2.0 Client Credentials Flow*. Spiegazione OAuth 2.0 Client Credentials Flow. URL: https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow (visitato il giorno 17/11/2025).

5.9 Verifica e validazione

L'ultimo periodo dello stage è stato dedicato alla fase di verifica e validazione di quanto prodotto. Tale pratica consiste nella scrittura ed esecuzione di *test di unità*, *test funzionali* e *test di integrazione* dedicati a verificare la copertura dei requisiti e il funzionamento del sistema. Il tempo a disposizione non è stato sufficiente alla scrittura di una suite completa di test, tuttavia sono stati scritti alcuni *test funzionali* dedicati a verificare il funzionamento dello schedulatore e il rispetto dei principali vincoli.

5.9.1 Test 1: ordine singolo

5.9.1.1 Scopo

Questo test ha lo scopo di validare la capacità di pianificare un singolo ordine di produzione. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori.

5.9.1.2 Implementazione

```
def test_single_order():
    expiration = (date.today() + timedelta(days=7)).isoformat()

    request_body = {
        "orders": [
            {
                "id": 1,
                "expiration_date": expiration,
                "number": "ORD-001",
                "pcs_ordered": 6,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 1,
                    "work_name": "ORD-001_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
```

```
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "02:00:00",
                        }
                    ],
                },
            }
        ],
        "printers": [
            {"id": 1, "name": "PRUSA-01"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)

    status, result = SchedulingService().schedule_with_batches(
        jobs=jobs,
        printers=printers,
        alg=ORToolsPrintingScheduler(),
    )

    assert status is True
    assert result.summary.total_jobs == 2
    assert result.summary.total_delay_minutes == 0
    assert result.printers["PRUSA-01"].job_count == 2
```

Codice 5.30: Implementazione del test ordine singolo

5.9.2 Test 2: ordini con materiali diversi

5.9.2.1 Scopo

Questo test ha lo scopo di validare la capacità di pianificare due ordini di produzione composti da prodotti stampati con due materiali differenti. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori e minimizzando il numero di cambi di materiale.

5.9.2.2 Implementazione

```
def test_materials_optimization():
    exp_pla = (date.today() + timedelta(days=4)).isoformat()
    exp_petg = (date.today() + timedelta(days=5)).isoformat()

    request_body = {
        "orders": [
            {
                "id": 10,
                "expiration_date": exp_pla,
                "number": "ORD-PLA",
                "pcs_ordered": 9,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 10,
                    "work_name": "ORD-PLA_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "01:00:00",
                        }
                    ],
                },
```

```
            },
            {
                "id": 11,
                "expiration_date": exp_petg,
                "number": "ORD-PETG",
                "pcs_ordered": 6,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 11,
                    "work_name": "ORD-PETG_WORK",
                    "printer_material": {"id": 2, "name": "PETG"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "01:30:00",
                        }
                    ],
                },
            },
        ],
        "printers": [
            {"id": 1, "name": "MK3S"},
            {"id": 2, "name": "BambuLab"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)

    status, result = SchedulingService().schedule_with_batches(
```

```
        jobs=jobs,
        printers=printers,
        alg=ORToolsPrintingScheduler(),
    )

    assert status is True
    assert result.summary.total_jobs == 5
    assert result.summary.total_delay_minutes == 0

    for printer_data in result.printers.values():
        if printer_data.job_count == 0:
            continue
        materials = {job.material for job in printer_data.jobs}
        assert len(materials) == 1
```

Codice 5.31: Implementazione del test ordini con materiali diversi

5.9.3 Test 3: ordine urgente

5.9.3.1 Scopo

Questo test ha lo scopo di validare la capacità di pianificare due ordini di produzione, di cui uno urgente. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori e minimizzando i ritardi.

5.9.3.2 Implementazione

```
def test_urgent_order():
    urgent_exp = (date.today() - timedelta(days=1)).isoformat()
    normal_exp = (date.today() + timedelta(days=7)).isoformat()

    request_body = {
        "orders": [
            {
                "id": 100,
```

```
                "expiration_date": urgent_exp,
                "number": "URGENT",
                "pcs_ordered": 2,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 100,
                    "work_name": "URGENT_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 2,
                            "time": "00:45:00",
                        }
                    ],
                },
            },
            {
                "id": 101,
                "expiration_date": normal_exp,
                "number": "NORMAL",
                "pcs_ordered": 6,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 101,
                    "work_name": "NORMAL_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "01:30:00",
                        }
                    ],
```

```
                },
            },
        ],
        "printers": [
            {"id": 1, "name": "OnlyOne"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)

    status, result = SchedulingService().schedule_with_batches(
        jobs=jobs,
        printers=printers,
        alg=ORToolsPrintingScheduler(),
    )

    assert status is True
    assert result.summary.total_jobs == 3
    assert result.summary.total_delay_minutes > 0

    jobs = sorted(
        result.printers["OnlyOne"].jobs, key=lambda j: j.start_time
    )
    assert jobs[0].order.number == "URGENT"
    assert jobs[0].delay_minutes == result.summary.total_delay_minutes
```

Codice 5.32: Implementazione del test ordine urgente

5.9.4 Test 4: ordine parziale

5.9.4.1 Scopo

Questo test ha lo scopo di validare la capacità di pianificare un ordine parziale. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori e calcolando correttamente i pezzi da stampare.

5.9.4.2 Implementazione

```
def test_partial_order():
    expiration = (date.today() + timedelta(days=6)).isoformat()

    request_body = {
        "orders": [
            {
                "id": 50,
                "expiration_date": expiration,
                "number": "PARTIAL",
                "pcs_ordered": 8,
                "pcs_printed": 5,
                "printer_work": {
                    "id": 50,
                    "work_name": "PARTIAL_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "01:00:00",
                        }
                    ],
                },
            }
        ],
```

```
        "printers": [
            {"id": 1, "name": "Prusa-Mini"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)

    status, result = SchedulingService().schedule_with_batches(
        jobs=jobs,
        printers=printers,
        alg=ORToolsPrintingScheduler(),
    )

    assert status is True
    assert result.summary.total_jobs == 1
    printer_data = result.printers["Prusa-Mini"]
    assert printer_data.job_count == 1
    job = printer_data.jobs[0]
    assert job.plate.pcs == 3
    assert job.order.get_remaining_pcs() == 3
```

Codice 5.33: Implementazione del test ordine parziale

5.9.5 Test 5: test vincoli di orario

5.9.5.1 Scopo

Questo test ha lo scopo di validare la capacità di rispettare i vincoli di orario. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori e rispettando i vincoli di orario.

5.9.5.2 Implementazione

```
def test_time_constraints():
    expiration = (date.today() + timedelta(days=30)).isoformat()
    request_body = {
        "orders": [
            {
                "id": 200,
                "expiration_date": expiration,
                "number": "TEST-VINCOLI-ORARI-WEEKEND",
                "pcs_ordered": 22,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 200,
                    "work_name": "TEST-VINCOLI-ORARI-WEEKEND_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 1,
                            "time": "03:00:00",
                        }
                    ],
                },
            }
        ],
        "printers": [
            {"id": 1, "name": "Stampante-Test"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
```

```
32      jobs =
        ↪ OrderConversionService().convert_orders_to_jobs(orders=orders)
33
34      status, result = SchedulingService().schedule_with_batches(
35          jobs=jobs,
36          printers=printers,
37          alg=ORToolsPrintingScheduler(),
38      )
39
40      assert status is True
41      assert result.summary.total_jobs == 22
42      assert result.summary.total_delay_minutes == 0
43
44      for job in result.printers["Stampante-Test"].jobs:
45          start: datetime = job.start_time
46          assert start.weekday() < 5
47          assert 9 <= start.hour < 18
```

Codice 5.34: Implementazione del test vincoli di orario

5.9.6 Test 6: test scelta del piatto

5.9.6.1 Scopo

Questo test ha lo scopo di validare la capacità di scegliere il piatto più adatto per minimizzare gli sprechi. Il test si considera superato se il modulo pianifica correttamente l'ordine senza errori e minimizzando gli sprechi.

5.9.6.2 Implementazione

```
1   def test_plate_choice():
2       expiration = (date.today() + timedelta(days=6)).isoformat()
3       request_body = {
4           "orders": [
5               {
```

```
                "id": 300,
                "expiration_date": expiration,
                "number": "PLATE-CHOICE",
                "pcs_ordered": 10,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 300,
                    "work_name": "PLATE-CHOICE_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 3,
                            "time": "01:00:00",
                        },
                        {
                            "id": 2,
                            "pcs": 5,
                            "time": "01:00:00",
                        },
                    ],
                },
            }
        ],
        "printers": [
            {"id": 1, "name": "Prusa-Plate-Test"},
        ],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)
```

```
38
39      status, result = SchedulingService().schedule_with_batches(
40          jobs=jobs,
41          printers=printers,
42          alg=ORToolsPrintingScheduler(),
43      )
44
45      assert status is True
46      assert result.summary.total_jobs == 2
47      assert result.summary.total_delay_minutes == 0
48
49      printer_data = result.printers["Prusa-Plate-Test"]
50      assert printer_data.job_count == 2
51      for job in printer_data.jobs:
52          assert job.plate.pcs == 5
```

Codice 5.35: Implementazione del test scelta del piatto

5.9.7 Test 7: test nessuna stampante disponibile

5.9.7.1 Scopo

Questo test ha lo scopo di gestire il caso in cui non vi sia alcuna stampante disponibile. Il test si considera superato se il modulo ritorna un errore.

5.9.7.2 Implementazione

```
1  def test_no_printer():
2      expiration = (date.today() + timedelta(days=5)).isoformat()
3
4      request_body = {
5          "orders": [
6              {
7                  "id": 70,
8                  "expiration_date": expiration,
```

```
                "number": "NOWAY",
                "pcs_ordered": 3,
                "pcs_printed": 0,
                "printer_work": {
                    "id": 70,
                    "work_name": "NOWAY_WORK",
                    "printer_material": {"id": 1, "name": "PLA"},
                    "plates": [
                        {
                            "id": 1,
                            "pcs": 1,
                            "time": "00:30:00",
                        }
                    ],
                },
            }
        ],
        "printers": [],
    }

    orders = [Order.model_validate(o) for o in request_body["orders"]]
    printers = [Printer.model_validate(p) for p in
    ↪  request_body["printers"]]
    jobs =
    ↪  OrderConversionService().convert_orders_to_jobs(orders=orders)

    try:
        status, _ = SchedulingService().schedule_with_batches(
            jobs=jobs,
            printers=printers,
            alg=ORToolsPrintingScheduler(),
        )
        assert status is False
    except Exception:
```

```
41          assert True
```

Codice 5.36: Implementazione del test nessuna stampante disponibile

5.9.8 Esecuzione dei test

Per eseguire i test è necessario aprire il terminale, navigare all'interno della cartella del progetto ed eseguire il comando `pytest`. I risultati sono i seguenti:

Test	Esito
test_materials_optimization.py	Passato
test_no_printer.py	Passato
test_partial_order.py	Passato
test_plate_choice.py	Passato
test_single_order.py	Passato
test_time_constraints.py	Passato
test_urgent_order.py	Passato

Tabella 5.1: Tabella dei risultati dei test Pytest.

Capitolo 6

Conclusioni

6.1 Copertura dei requisiti

La seguente tabella riporta la copertura dei requisiti, ovvero un resoconto completo dei requisiti soddisfatti e non soddisfatti.

Identificativo	**Soddisfatto**
RO0	Sì
RO1	Sì
RO2	Sì
RO3	Sì
RO4	Sì
RO5	Sì
RO6	Sì
RO7	Sì
RO8	Sì
RO9	Sì
RO10	Sì
RO11	Sì
RO12	Sì
RD0	Sì
RD1	Sì
RD2	No
RD3	No
RD4	Sì
RD5	Sì
RF0	No
RF1	No
RF2	No

Tabella 6.1: Tabella del tracciamento del soddisfacimento dei requisiti.

6.2 Analisi del prodotto

In questa sezione viene presentata un'analisi critica del prodotto consegnato all'azienda ospitante.

6.2.1 Prodotto

Il prodotto sviluppato rappresenta sicuramente un buon contributo all'automazione della gestione degli ordini di produzione. Il sistema è stato integrato correttamente con il software gestionale esistente ed è quindi immediatamente utilizzabile dall'azienda. È possibile inoltre continuare a estendere e a migliorare il codice, in quanto sono stati adottati *design pattern* specifici per aumentare la manutenibilità. Uno dei punti critici dello schedulatore è sicuramente rappresentato dalle sue prestazioni, in quanto il sistema impiega molto tempo per il calcolo di una soluzione ottima. L'implementazione del *batching* riesce a mitigare il problema, a patto che una soluzione sub-ottima sia sufficiente per l'utente. Non è stato inoltre possibile, per mancanza di tempo, valutare la differenza media tra una soluzione ottima e una soluzione sub-ottima.

6.2.1.1 Miglioramenti

Il prodotto consegnato può essere migliorato nei seguenti modi:

- Implementazione dei requisiti desiderabili e facoltativi mancanti;
- Miglioramento delle performance tramite ottimizzazione del numero di variabili generate.

Sitografia

Documentazione ufficiale Conventional Commits 1.0.0. URL: https://www.conventionalcommits.org/en/v1.0.0/ (cit. a p. 6).

Framework Scrum. URL: https://it.wikipedia.org/wiki/Scrum_(informatica) (cit. a p. 7).

Framework Scrum. Licenza: CC BY-SA 4.0; l'immagine SVG originale è stata convertita in PNG per l'inclusione nel documento. URL: https://commons.wikimedia.org/wiki/File:Scrum_process.svg (visitato il giorno 23/10/2025).

Kanban board. Licenza: CC BY-SA 4.0; l'immagine SVG originale è stata convertita in PNG per l'inclusione nel documento. URL: https://commons.wikimedia.org/wiki/File:Abstract_Kanban_Board.svg (visitato il giorno 25/10/2025).

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/ (cit. a p. 7).

OAuth 2.0 Client Credentials Flow. Spiegazione OAuth 2.0 Client Credentials Flow. URL: https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow (visitato il giorno 17/11/2025) (cit. a p. 66).

OpenAPI. Sito ufficiale documentazione OpenAPI. URL: https://swagger.io/specification/ (visitato il giorno 08/11/2025) (cit. a p. 25).

Pattern a layer. Spiegazione pattern a layer. URL: https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensive-guide-1e2j#1-separation-of-concerns (visitato il giorno 08/11/2025).

Pydantic. Sito ufficiale Pydantic. URL: https://docs.pydantic.dev/latest/ (visitato il giorno 08/11/2025) (cit. a p. 25).

Python. Sito ufficiale Python. URL: https://www.python.org/ (visitato il giorno 08/11/2025) (cit. a p. 25).

Repository pattern. Spiegazione repository pattern. URL: https://www.geeksforgeeks.org/system-design/repository-design-pattern/ (visitato il giorno 13/11/2025).

Sito Mugalab. URL: https://mugalab.com/ (cit. a p. 1).

Sito Spazio Dev. URL: https://spaziodev.eu/ (cit. a p. 1).

Sito ufficiale Git. URL: https://git-scm.com/ (cit. a p. 5).

Sito ufficiale Gitea. URL: https://about.gitea.com/ (cit. a p. 5).

Sito ufficiale Plane. URL: https://plane.so/ (cit. a p. 8).

Spiegazione completa Gitflow workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (cit. a p. 5).

Telegram. Sito ufficiale Telegram. URL: https://telegram.org (visitato il giorno 23/10/2025) (cit. a p. 9).