Norme di progetto

v1.4.0

Farmacode

farmacode.swe.unipd@gmail.com

Registro delle modifiche

Versione	Data	Scrittori	Revisori	Descrizione
1.4.0	2024-04-04	Carraro Alessandro		Aggiunte MVP: git actions e altro
1.3.0	2024-04-03	Carraro Alessandro		Ristrutturazione documento
1.2.0	2024-02-29	Favaron Riccardo	Bomben Filippo	Migliorie a sezioni 4.1.4.3 e aggiunta delle nuove sezioni 3.1.2, 3.1.3, 3.2.3.2
1.1.1	2024-02-28	Favaron Riccardo	Bomben Filippo	Migliorie a sezioni 3.1.5, 3.1.5.1, 4.1.3.2
1.1.0	2024-02-27	Favaron Riccardo	Bomben Filippo	Migliorie a sezioni 1.5, 2.1.2, 2.1.3, 2.1.4, 3.1.7
1.0.0	2024-02-13	Rosson Lorenzo	Baggio Matteo	Verifica e revisone del documento
0.14.0	2024-02-04	Pandolfo Mattia	Bomben Filippo	Aggiunta sezione metriche per la qualità
0.13.0	2024-01-14	Baggio Matteo Rosson Lorenzo	Bomben Filippo	Apportate alcune migliorie e correzioni sezione 3.1
0.12.0	2023-12-22	Passarella Alessandro	Carraro Alessandro	Stesura sezione 1.4
0.11.1	2023-12-22	Carraro Alessandro	Passarella Alessandro	Completata sezione 3.1.7 Struttura file di progetto
0.11.0	2023-12-21	Carraro Alessandro	Passarella Alessandro	Inizio sezione 3.1.7 Struttura file di progetto
0.10.0	2023-12-21	Carraro Alessandro	Passarella Alessandro	Completata sezione 3.1.4 Norme tipografiche
0.9.1	2023-12-20	Rosson Lorenzo	Passarella Alessandro	Correzione sezione use case
0.9.0	2023-12-14	Bomben Filippo	Baggio Matteo	Stesura documento da sezione 3.3 a 3.7, eccetto 3.4.2.2
0.8.0	2023-12-10	Rosson Lorenzo	Bomben Filippo	Completata prima stesura della sezione 2, e modificate alcune parti della sezione 3
0.7.1	2023-12-05	Rosson Lorenzo	Bomben Filippo	Completata sezione 4 con alcuni miglioramenti
0.7.0	2023-12-02	Rosson Lorenzo	Favaron Riccardo	Realizzata prima stesura sezione 4
0.6.0	2023-11-25	Rosson Lorenzo	Favaron Riccardo	Realizzata prima stesura sezione 3.2, apportate modifiche alla sezione 3.1.5
0.5.0	2023-11-21	Baggio Matteo	Carraro Alessandro	Trasferimento da LaTeX a Typst del documento
0.4.0	2023-11-20	Passarella Alessandro	Carraro Alessandro	Completamento stesura sezione 3.1
0.3.0	2023-11-18	Baggio Matteo	Carraro Alessandro	Completamento stesura sezione 1
0.2.0	2023-11-15	Baggio Matteo Passarella Alessandro	Carraro Alessandro	Stesura indice
0.1.0	2023-11-12	Bomben Filippo Rosson Lorenzo	Baggio Matteo	Stesura iniziale del documento

Indice

1) Introduzione

1.1) Scopo del documento

Questo documento è stato creato per identificare le *best practices* di progetto e per stabilire la *way of working*, ovvero una metodologia di lavoro chiara, nel corso dell'attività produttiva. L'obiettivo è garantire una gestione omogenea e coesa del lavoro. Per facilitare il monitoraggio del progresso e consentire un approccio incrementale è consentito apportare modifiche e aggiornamenti alle pratiche successivamente descritte, in modo di addattarsi a nuove attività e/o strumenti di lavoro, il tutto viene registrato nello storico delle modifiche del documento.

1.2) Scopo del prodotto

Al giorno d'oggi l'ambito degli *e-commerce* si sta sempre più espandendo ed evolvendo. La presenza di negozi virtuali permette di accedere a molti dati legati agli acquisti, alle preferenze ed al comportamento degli utenti. Questi dati se analizzati propriamente permettono di prevedere preferenze e comportamenti futuri degli utenti, dando spazio ad operazioni di marketing mirate. Il progetto ha lo scopo di realizzare un *sistema di raccomandazione* con relativa interfaccia web che guidi le attività dell'azienda, utilizzatrice del prodotto finale, suggerendo a quali clienti rivolgere le singole attività di marketing e commerciali, cercando i migliori clienti target a cui indirizzare determinati prodotti.
L'appicazione è sviluppata sotto forma di *webapp* per la sua comodità, favorendo così l'accesso e la fruizioneda diversi dispositivi, sistemi o browser.
Dall'interfaccia utente del sistema software sarà possibile selezionare uno specifico cliente e visualizzare i prodotti da lui acquistati e quelli che il sistema ha individuato come raccomandati. Inoltre selezionato un articolo o un insieme di articoli il sistema suggerisce a quali clienti proporli, selezionandoli in base a quanto probabile siano interessati per i prodotti analizzati. I vari prodotti possono essere filtrati per categoria così da facilitare ricerche e restringere il campo di soluzione.
Ogni risultato restituito dal sistema di raccomandazione è classificabile tramite un feedback così da poter eventualmente correggere il tiro dell'*algoritmo* che ha fornito l'esito della suggerimento. L'utente amministratore ha la possibilità di creare ulteriori account per eventuali operatori che necessitano di utilizzare l'applicativo.

1.3) Glossario

Al fine di evitare eventuali equivoci o incomprensioni riguardo la terminologia utilizzata all'interno di questo documento, si è deciso di adottare un Glossario, con file apposito, in cui vengono riportate tutte le definizioni rigogliose delle parole ambigue utilizzate in ambito di questo progetto. Nel documento appena descritto verranno riportati tutti i termini definiti nel loro ambiente di utilizzo con annessa descrizione del loro significato.
La presenza di un termine all'interno del Glossario è evidenziata dal *colore blu*.

1.4) Miglioramenti e maturità del documento

Questo documento è stato creato seguendo un approccio incrementale, il che implica la sua natura adattabile e suscettibile di modifiche nel tempo. Queste modifiche saranno apportate in risposta alle esigenze concordate tra i membri del gruppo e il proponente. Pertanto, questa versione del documento non deve essere considerata come una versione definitiva o completa, ma piuttosto come un punto di partenza che sarà ulteriormente sviluppato e aggiornato per meglio rispondere alle mutevoli esigenze del progetto.

1.5) Riferimenti

1.5.1) Riferimenti normativi

- *Capitolato* C2 - Sistema di raccomandazione
  https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C2.pdf;
- Regolamento progetto ditattico
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf.

1.5.2) Riferimenti informativi

Documentazione delle principali tecnologie utilizzate

- Documentazione GitHub
  https://docs.github.com/en;
- Documentazione Typst
  https://typst.app/docs/;

Slide corso di ingegneria del software

- T2 - Processi di ciclo di vita
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T2.pdf;
- T6 - Progettazione software
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf;
- T7 - Qualità del software
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T7.pdf;
- T9 - Verifica e validazione: introduzione
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T9.pdf;
- T10 - Verifica e validazione: analisi statica
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T10.pdf;
- T11 - Verifica e validazione: analisi dinamica
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T11.pdf;
- P2 - Diagrammi dei casi d'uso
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf.

Slide corso Metodi e Tecnologie per lo Sviluppo Software
Le seguenti dispense sono relative all'anno accademico 2022/2023.

- 2 - Issue Tracking System
  https://stem.elearning.unipd.it/pluginfile.php/478029/mod_resource/content/0/2-IssueTrackingSystem_22_23_libre.pdf;
- 3 - GitHub Issue Tracker
  https://stem.elearning.unipd.it/pluginfile.php/482498/mod_resource/content/0/3-GitHubIssueTracker.pdf;
- lab1 - GitHub Issue Tracker
  https://stem.elearning.unipd.it/pluginfile.php/482500/mod_resource/content/1/GitHubIssueTracker.pdf;

- 4 - Git
  https://stem.elearning.unipd.it/pluginfile.php/490005/mod_resource/content/0/4-GIT.pdf;
- lab2 - GitHub vcs
  https://stem.elearning.unipd.it/pluginfile.php/494455/mod_resource/content/0/lab2-github_vcs.pdf.

2) Processi primari

2.1) Fornitura

2.1.1) Descrizione e Scopo

In questa sezione sono elencate tutte le norme che il gruppo è tenuto ad osservare e rispettare per garantire il mantenimento di un rapporto utile e il più trasparente possibile con il proponente e i committenti per l'intera durata del progetto.
Il processo di fornitura definisce l'insieme di attività, compiti e risorse necessari al fornitore per portare a termine con successo il progetto. Il suo obiettivo principale è tracciare e descrivere le attività svolte dai membri del team di sviluppo. Questo tracciamento consente di valutare il lavoro completato, quantificare ciò che ancora deve essere realizzato e confrontare gli avanzamenti con le richieste del proponente, fornendo così una "istantanea" costante dello stato dei lavori e del bilancio di progetto.
Durante questa fase, il gruppo è tenuto a stabilire i contatti con il proponente e a definire i requisiti per il *MVP* (Minimum Viable Product) da concordare con quest'ultimo, basandosi su tempistiche, costi e importanza (intesa come incidenza sulla qualità del prodotto).

2.1.2) Rapporti con il proponente

Il team si impegna a mantenere contatti costanti con il proponente per l'intera durata del progetto. C'è la volontà di rapportarsi di settimana in settima, tuttavia non si è riuscitti ad accordarsi per fissare un giorno ed un orario preciso, causa trasferte da parte del proponente. Comunque massima disponibilità da entrambe le parti. Questo impegno non solo assicura in parte il corretto sviluppo dei lavori, ma facilita anche lo scambio di informazioni e feedback tra le parti, contribuendo a garantire il rispetto degli accordi stabiliti.
Gli incontri con la proponente è attuata con l'utilizzo di canali diversi, quali e-mail, Google Meet e Zoom.
Gli incortri possone essere anche richiesti per le seguenti capacità:

- Chiarimenti relativi ai vincoli o requisiti del capitolato;
- Chiarimenti relativi alle tecnologie utilizzate e/o proposte;
- Richieste di formazione su specifiche nuove tecnologie e/o concetti;
- Richieste di feedback su quanto realizzato.

Per ogni incontro verrà redatto un verbale esterno, il quale riporterà sul proprio nome ed all'interno del documento stesso la data dell'incontro. Nella sezione comunicazioni e nella sezione incontri è possibile consultare le norme e modalità con le quali avvengono questi contatti.

2.1.3) Documentazione fornita

In questa sezione viene presentanto un elenco contenente tutta la documentazione che deve venire fornita a proponente e/o committente, per obblighi accademici o per volontà del team.

- **Valutazione dei Capitolati**

  Il documento `Valutazione dei Capitolati` contiene l'analisi dei vari capitolati d'appalto proposti, con l'obiettivo di individuare quello più adatto al gruppo farmacode.
  Per ogni capitolato la struttura è la seguente:
  - Descrizione: comprende il nome dell'azienda proponente, i committenti e una breve descrizione dell'obiettivo di progetto;
  - Dominio applicativo: descrive più nel dettaglio il contesto del progetto;

- Dominio tecnologico: descrive le tecnologie proposte e/o da utilizzate per lo sviluppo;
- Aspetti positivi: elenco degli aspetti positivi riguardanti il progetto;
- Aspetti critici: elenco degli aspetti critici riguardanti il progetto;
- Consclusioni: descrizione delle motiviazioni di scelta o non scelta del capitolato valutato.

- **Analisi dei Requisiti**

  Documento `Analisi dei Requisiti` contenente lo studio dei requisiti del prodotto software, loro classificazione, scenario e gestione. Definisce quindi le funzionalità che il prodotto offre e i requisiti da soddisfare affinchè si realizzi un output conforme alle rischieste fatte dal proponente. La struttura del documento è descritta nella seziona contenuti relativa.

- **Piano di Progetto**

  Documento `Piano di Progetto` contenente la pianificazione delle attività di progetto, la gestione dell'organizzazione ruolistica ed il bilancio del progetto. È soggetta a versionamento e approvazione, inoltre è redatta dal Resonsabile a quel momento con l'aiuto degli Amministratori. La struttura del documento è descritta nella seziona contenuti relativa.

- **Piano di Qualifica**

  Documento `Piano di Qualifica` contenente le normative e metriche riguardanti i processi di qualifica in adozione dal gruppo per garantire la qualità del prodotto che si sta sviluppando. È uno strumento essenzaiale per la gestione del processo di sviluppo software in quanto garantisce che il prodotto finale sia conforme ai requisiti e alle aspettarive del committente. La struttura del documento è descritta nella seziona contenuti relativa.

- **Glossario**

  Documento `Glossario` utile per favorire una corretta consultazione della documentazione di progetto. È destinato sia ai membri del gruppo di progetto, sia all'azienda proponente, che ai committenti Viene accompagnato da una sua versione online disponibile nel sito vetrina del gruppo al seguente link. La struttura del documento è descritta nella seziona contenuti relativa.

- **Lettera di presentazione**

  Documento `Lettera di Presentazione` accompagna e introduce i documenti e il prodotto sowftare sviluppato in ogni fase di revisione di progetto. Illustra le risorse fornite al committente e all'azienda proponente.

2.1.4) Strumenti

Segue un elenco degli strumenti utilizzati dal team per il processo di fornitura:
- *Typst* per la stesura dei documenti ufficiali di progetto;
- Suite prodotti Google (drive, documenti e fogli), per la condivisione di note, appunti, tabelle e grafici di funzione solitamente di tipo organizzativo;
- Google Meet per effettuare videoconferenze con l'azienda proponente;
- Zoom per effettuare videoconferenze con i committenti di progetto e con l'azienda proponente;
- *GitHub* per l'hosting e versionamento del prodotto software (documentazione inclusa), per gestire le issue di progetto;
- *Diagrams.net* (draw.io) per la realizzazione grafica dei *casi d'uso*
- *KeyNote* per la realizzazione delle slide dei diari di bordo.

2.2) Sviluppo

2.2.1) Descrizione e Scopo

In questa sezione del documento, vengono definite le linee guida e le norme essenziali atte a dirigere le attività di sviluppo in modo accurato e uniforme.
L'obiettivo principale è garantire una coerenza completa nei metodi utilizzati, mirando al contempo a promuovere il raggiungimento di una qualità superiore nel prodotto finale.
Questa sezione svolge un ruolo cruciale nel plasmare il processo di sviluppo, fornendo una struttura chiara e definita per le operazioni, che permette di mantenere standard solidi in tutte le fasi del ciclo di vita del progetto.
La sua importanza sta nel contribuire alla creazione di un ambiente di lavoro orientato al raggiungimento di un risultato finale che soddisfi le aspettative e rispetti gli standard predefiniti.
Il processo di sviluppo di un prodotto software è suddivisibile come segue:

- Analisi dei requisiti;
- Design architetturale;
- Design del software;
- Programmazione e verifica;
- Integrazione.

2.2.2) Analisi dei requisiti

Il processo di analisi dei requisiti si colloca come prima fase dello sviluppo software, un momento cruciale in cui è imperativo delineare con precisione e robustezza gli scenari dei casi d'uso e le relative necessità o requisiti del sistema. Questa fase è di fondamentale importanza, perché implica la comprensione approfondita delle richieste degli utenti e degli *stakeholder*, offrendo una base solida per il progresso del ciclo di sviluppo.
Nel corso dell'analisi dei requisiti, il focus è posto sull'identificazione, la documentazione e la comprensione esaustiva delle funzionalità necessarie che il sistema dovrà incorporare.
Inoltre, durante l'analisi dei requisiti, è essenziale stabilire una comunicazione efficace con il proponente, al fine di garantire che tutte le prospettive e le esigenze rilevanti siano adeguatamente considerate.

Casi d'uso

I *Casi d'uso* rappresentano azioni o sequenze di azioni collocabili in specifici scenari, caratterizzate da una richiesta e da una risposta. La loro definizione e realizzazione deve seguire gli standard imposti dal linguaggio UML, ed alcune regole interne al progetto.
Ogni caso d'uso deve essere così composto:

- identificazione e nomenclatura: il formato concordato è il seguente:

  UC.x.y - Nome

  dove: x, rappresenta il numero identificativo del caso d'uso generico; y, rappresenta l'id del sotto caso relativo, che può essere composto da più cifre (n.n.n.n …). E "Nome" rappresenta il nome del caso d'uso da attribuire in modo chiaro e consono;
- figura;
- attori: utente o componente che può svolgere quella determinata azione o chiedere quel servizio;
- precondizioni: condizioni di utente e/o sistema, necessarie affinchè si verifichi quel caso d'uso/scenario;

- postcondizioni: rappresentano lo stato del sistema e/o utente, dopo che il caso d'uso è stato eseguito;
- scenario principale: in questa sezione si elencano le fasi che caratterizzano il caso d'uso;
- estensioni (se presenti): in questa sezione vanno elencate eventuali estensioni, nel caso ci possano essere degli scenari alternativi;
- generalizzazioni (se presenti): in questa sezione vanno specificati ed elencati i possibili sotto casi d'uso, e la presenza o meno di una generalizzazione tra gli attori coinvolti.

Linee guida interne

- **Attore:**

Figura 1: Attore

Un attore è un elemento esterno che interagisce con il sistema. Gli attori possono assumere diverse forme, come utenti, persone, macchine, altri sistemi informatici o organizzazioni. Un caso d'uso definisce una specifica funzionalità fornita agli attori senza specificarne dettagli implementativi. Nel diagramma dei casi d'uso, gli attori sono rappresentati da figure umane stilizzate, ciascuno identificato da un'etichetta contenente il suo nome.

- **Caso d'uso**

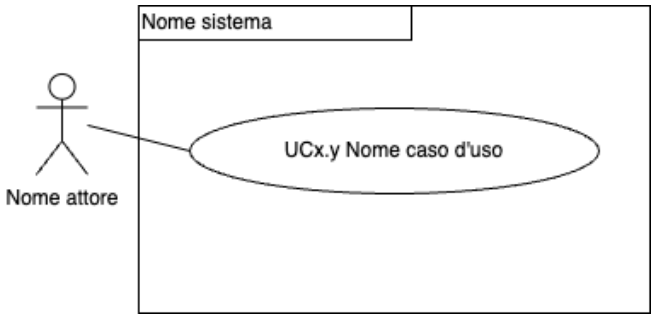

Figura 2: Caso d'uso

I casi d'uso rappresentano le azioni o funzionalità che gli utenti possono eseguire all'interno di un sistema. Ogni caso d'uso è collegato mediante una linea continua agli attori che hanno accesso a quella specifica funzionalità, fornendo una chiara relazione tra gli utenti e le azioni che possono compiere nel sistema software.

- **Inclusione**

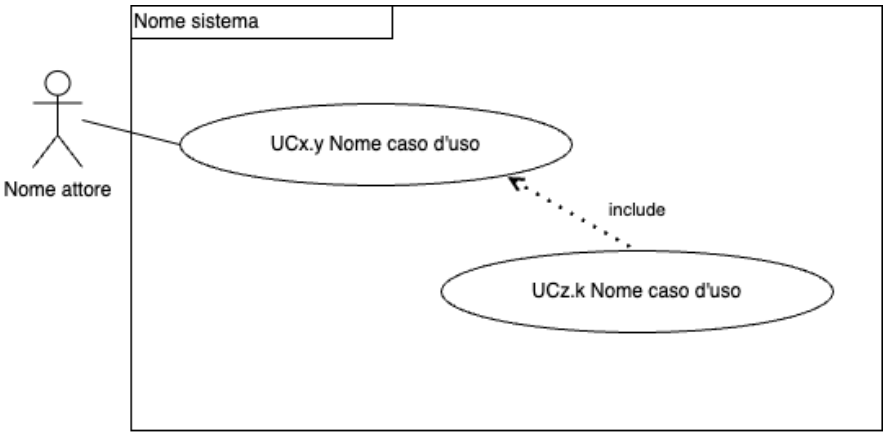

Figura 3: Inclusione

L'include serve a delineare un UC che fa parte sempre dello stesso scenario principale, solitamente viene usato quando uno stesso UC viene sempre utilizzato da più UC simultaneamente.

- **Estensione**

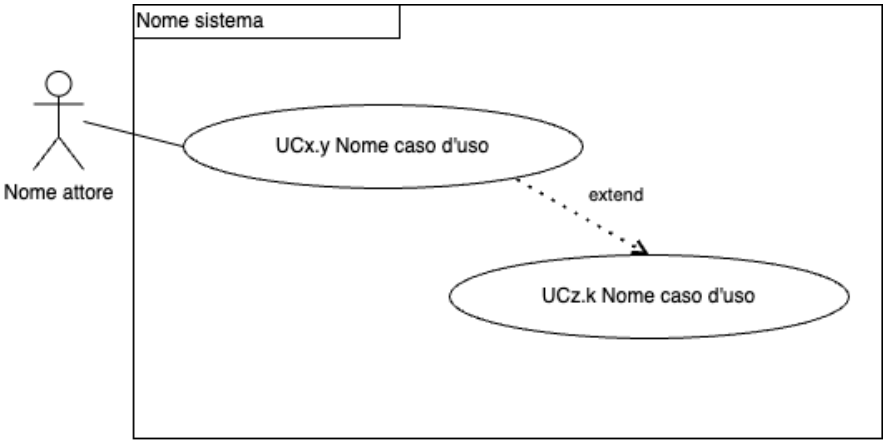

Figura 4: Estensione

L'extend serve a delineare un UC che ha le stesse precondizioni dell'UC da cui è derivato, ma ha postcondizioni differenti perchè si verifica un evento che porta a deviare dallo scenario principale ad uno scenario alternativo: esempi sono scenari di errore.

- **Generalizzazione**

  1. Generalizzazione casi d'uso:

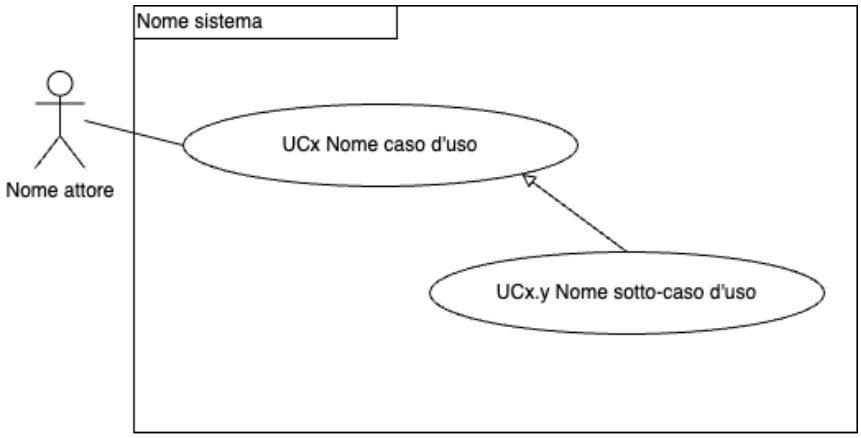

Figura 5: Generalizzazione casi d'uso

La generalizzazione serve a specificare meglio una funzionalità e coincide con l'OR esclusivo;

1. Sotto-UC: Se si sceglie di specificare meglio una funzionalità non tramite generalizzazioni ma tramite Sotto-UC, la funzionalità deve rimanere invariata nei Sotto-UC (Ad esempio: "Visualizzazione oggetto" scomposto in: "Visualizzazione nome oggetto", …). Utilizzare i sotto-UC coincide con l'AND logico;

2. Generalizzazione attori:

Figura 6: Generalizzazione attori

La generalizzazione tra attori serve per evidenziare ed identificare le funzionalità, ovvero i casi d'uso, disponibili per l'attore "A", che sono anche utilizzabili dall'attore "B" (non vale il viceversa).

Requisiti

I requisiti devono possedere un identificativo, composto come segue:

R[Importanza][Tipologia] X

Dove:

- Importanza: Indica il grado di importanza del requisito ed indirettamente la sua incidenza sul progetto. Un requisito può essere:

  1. O –> "Obbligatorio";

     Un requisito deve essere defenito ed identificato come obbligatorio se ritenuto tale dal proponente, o se considerato tale e soddisfabile, dopo una cauta ed approfondita analisi, dal team di sviluppo.

  2. D –> "Desiderabile";

Un requisito deve essere defenito ed identificato come desiderabile, se ritenuto tale dal gruppo, in accordo con il proponente. Un requisito di questa categoria è da considerarsi soddisfabile in un secondo momento, a seconda di tempistche e costi.

3. OPT –> “Opzionale”.

   Un requisito deve essere defenito ed identificato come opzionale, se ritenuto tale dal gruppo, in accordo con il proponente. Un requisito di questa categoria è da considerarsi di valore aggiunto per il prodotto, anche se secondario, ma insoddisfabile date tempistiche e costi.

- Tipologia: Indica il tipo di requisito in esame. Un requisito può essere:

  1. F –> “Funzionale”;

     I requisiti funzionali descrivono le funzionalità del sistema, le azioni che il sistema può compiere e le informazioni che questo può fornire.

  2. Q –> “di Qualità”;

     I requisiti di qualità descrivono come un sistema deve essere, o come il sistema deve esibirsi, per soddisfare le esigenze dell’utente.

  3. V –> “di Vincolo”;

     I requisiti di vincolo descrivono i limiti e le restrizioni che un sistema deve rispettare per soddisfare le esigenze dell’utente.

  Sono state successivamente concordate altre tre tipologie ritenute dal team, in seguito ad un’attenta analisi, di secondaria importanza data la natura del capitolato. Questi ultimi requisiti non saranno accompagnati da un identificativo, ma verranno posti nelle sezioni apposite in modo da non perderne la tracciabilità.

1. “d’Ambiente”;

   I requisiti d’ambiente si riferiscono alle condizioni e alle risorse necessarie per sviluppare, testare e implementare il software in un ambiente operativo specifico. Questi requisiti forniscono le specifiche riguardanti l’infrastruttura tecnologica e le configurazioni d’ambiente.

2. “di Performance”;

   I requisiti di performance definiscono le prestazioni e le caratteristiche di rendimento che il sistema deve raggiungere per soddisfare le aspettative degli utenti e del proponente.

3. “di Sicurezza”.

   I requisiti di sicurezza delineano le misure di sicurezza e i comportamenti attesi per proteggere il sistema da minacce esterne o interne.

2.2.3) Progettazine

L’attività di progettazione segue ed utilizza la fase di analisi dei requisiti per definire ancor più struttura, vincoli e specifiche tecniche del prodotto software in oggetto.
La fase di progettazione mira inoltre a facilitare definizione, suddivisione e quindi pianificazione delle attività di codifica del prodotto, beneficiando, se eseguita in modo corretto e vantaggioso, il ciclo di vita del software.
La progettazione inizia con la creazione di un *PoC* (Proof of Concept), un prodotto software solitamente usa e getta, che mira a dimostrare la fattibilità del progetto. Durante questa prima

fase, vengono scelte le tecnologie da adottare e viene abbozzata una prima struttura del prodotto, andando a definirne le parti, sempre con l'ausilio dei casi d'uso e relativi requisiti, analizzati nella fase precedente. Vengono infine identificate e concordate con il proponente le funzionalità considerate di maggiore importanza da sviluppare in questa prima bozza del prodotto.
Successivamente, durante la fase adibita allo sviluppo di un MVP, verranno svolti studi più approfonditi sull'architettura del software in modo da migliorarne la qualità e manutenibilità generale. Tali studi verranno raccolti in un relativo documento allegato alla revisone di *PB* (Product Baseline).

2.2.4) Progettazioen PoC

Bozza di Architettura

Le scelte concordate per la realizzazione del PoC sono le seguenti: il prodotto è suddivisibile in 5 Layer principali:

- Livello Dati: comprendente il Database relazionale, contente parte del dataset fornitoci dal proponente;
- Livello di Elaborazione: comprendente gli script Python adibiti alla gestione e applicazione, del algoritmo di raccomandazioni;
- Livello di Logica: è formato dalle *API* che permetteranno la comunicazione tra Webapp ed algoritmo, e dalla gestione di quest'utlime tramite script PHP;
- Livello di Presentazione: è composto dalla Webapp, che permetterà di utilizzare il prodotto al utente finale.

Algoritmo di raccomandazione

L'approccio concordato con il proponente, per quanto riguarda l'algoritmo di raccomandazione, prevede l'adozione di una strategia singolare basata su ratings espliciti, lasciando la possibilità di esplorare un approccio misto durante lo sviluppo della versione di MVP del prodotto.
Confrontandoci con il proponente si è deciso di utilizzare la libreria *Surprise* di python, la quale contiene e permette di usare vari algoritmi di predizioni, moduli di predizione e vari modelli di allenamento. Al seguito di vari test di performance si è deciso di utilizzare come algoritmo l'algoritmo di predizione *SVD* (Singular Value Decomposition), il quale non richiede nessun modulo di predizione.
Per quanto riguarda il modello di allenamento, si è invece scelto di utilizzare il KFold, un semplice iteratore che si basa sulla cross-validation.
I dati forniti erano grezzi per questo, prima di farli elaborare dallo script, si è dovuto strutturarli nel seguente modo sensato: abbiamo creato un file .csv contenente solo tuple di tipo "Utente;Oggetto;Rating", dove il rating è il prodotto della manipolazione del numero di volte le quali un oggetto è stato comprato da un utente. Per non penalizzare troppo oggetti comprati meno volte o agevolare troppo pochi prodotti acquistati un numero considerevole di volte, abbiamo trattato il valore di rating come argomento di un logaritmo.

Tecnologie scelte

Segue un elenco delle tecnologie scelte ed adottate per lo sviluppo del PoC:

- Configurazione:
  1. *Docker*, per la gestione della configurazione;
- Livello Dati:
  1. Mysql per il datatbase;
- Livello Elaborazione:
  1. Python, come linguaggio per il sistema di raccomandazioni;
  2. *Numpy*, surprise e *pandas*, come librerie.

- Livello di Logica:
  1. *PHP*.
- Livello di Presentazione:
  1. *React*.

Programmazione e verifica del software
In questa sezione sono raccolte tutte le norme e regole che i programmatori in carico sono tenuti ad osservare durante il processo di codifica. La programmazione e relativa verifica, è una fase fondamentale, durante la quale chi ne è incaricato, inizia a plasmare e implementare il prodotto che l'utente finale andrà ad utilizzare.

- Linguaggi e ambiente:
  Per lo sviluppo del prodotto il team userà vari linguaggi di programmazione a seconda di esigenze e vincoli, imposti sia dal capitolato che dal proponente. Se ne riporta qui sotto un elenco per una più facile consultazione:
  1. Python, per l'agortimo di raccomandazione, e uso di sue svariate librerie come surprise, panda e numpy;
  2. Mysql per la realizzazione del database relazionale;
  3. Php per il backend della webapp;
  4. React e Node.js per il frontend dell'interfaccia web.

  Allo scopo di avere un ambiente coeso ed organizzato i componenti del gruppo sono tenuti ad utilizzare la configurazione creata appositamente Docker durante lo sviluppo. Docker è un software che permette di creare container per semplificare la gestione e la distribuzione di librerie e pacchetti e creare ambienti di lavoro uniformati.

- Stile di codifica:
  Al fine di poter lavorare in un ambiente omogeneo ed ordinato il team ha deciso di predisporre delle automazioni che migliorino la qualità del codice ed organizzino la sua formattazione.
  1. Python: Si è deciso di appoggiarsi a ruff, un sistema automatico di formattazione e analisi statica del codice. Esso è integrato nella repository tramite una GitHub action.

- Lunghezza e complessità:
  Le funzioni e i metodi integrati nel codice del prodotto devono aderire rigorosamente agli standard di qualità stabiliti nel contesto del progetto. La filosofia che ogni membro del team si impegna ad adottare si concentra sull'incoraggiare il riuso del codice, sulla facilità del suo mantenimento e sull'ottimizzazione delle prestazioni.
  Segue un elenco dei principali principi guida:

  1. Riuso del Codice: Ogni componente del gruppo è incoraggiato a sviluppare funzioni e metodi modulari che possano essere riutilizzati in diverse parti del progetto, sfavorendo la duplicazione del codice, e facilitando la manutenzione e la gestione delle funzionalità comuni.

  2. Mantenimento del Codice: Le funzioni devono essere scritte in modo chiaro e documentate adeguatamente per facilitare il mantenimento del codice nel tempo. Commenti significativi e una documentazione chiara aiutano i membri del team a comprendere rapidamente le funzionalità e apportare modifiche quando necessario.

  3. Efficienza del Codice: Si presta particolare attenzione all'efficienza del codice. Le funzioni dovrebbero essere progettate in modo ottimale per garantire un'esecuzione efficiente.

L'attenzione è rivolta alla complessità algoritmica, all'utilizzo appropriato delle risorse e alla minimizzazione di operazioni computazionalmente costose.

4. Testabilità: Ogni funzione dovrebbe essere progettata in modo tale da essere facilmente testabile. L'ideale sarebbe seguire un approccio *TDD* (Test Driven Development), quando e quanto più possibile.

Integrazione
La sezione dedicata all'integrazione del sistema e del software delinea il processo mediante il quale diverse componenti, moduli o servizi del progetto vengono combinati e testati per formare un sistema funzionante e coeso. L'obiettivo primario di questo processo è garantire che tutte le parti del sistema operino in armonia, soddisfacendo i requisiti funzionali e di prestazione stabiliti. Il team si impegna fin da subito nell'integrare le varie componenti del prodotto, in modo solido e al contempo elastico, in modo che quanto esplorato durante la produzione del PoC possa essere utile una volta inziati i lavori sul prodotto di MVP.

2.3) Gestione operativa

2.3.1) Descrizione e Scopo

Questa sezione fornirà dettagli sull'installazione del software come i requisiti di sistema e le procedure necessarie per eseguire correttamente il prodotto. Inoltre illustrerà una guida all'utilizzo, esponendo le principali funzionalità utilizzabili ed il come interagire con il sistema.
Per quanto riguarda il PoC si rimanda al relativo README.md disponibile nel repository.

2.3.2) Utilizzo operativo

In questa sezione verranno descritti i processi di installazione, erogazione ed utilizzo del prodotto.

2.4) Manutenzione

2.4.1) Descrizione e Scopo

Questa sezione fornisce una panoramica delle pratiche di manutenzione adottate dal team durante lo sviluppo del prodotto software, delineandone gli obiettivi e lo scopo principale. Include informazioni sulle attività svolte per garantire la stabilità, l'efficienza e la sicurezza del software nel tempo.

2.4.2) Correzione

La correzione si concentra sull'identificazione e sulla risoluzione di difetti o problemi nel software. Questa sezione descrive il processo di individuazione degli errori, la loro classificazione e la successiva implementazione delle correzioni. Si illustrano anche le pratiche adottate per garantire che le correzioni siano tempestive e accurate, contribuendo così a migliorare la qualità complessiva del prodotto.

2.4.3) Adattamento

Durante le successive implementazioni del software di progetto verranno descritte le operazioni di adattamento.

2.4.4) Evoluzione

Durante le successive implementazioni del software di progetto verranno descritte le operazioni di evoluzione.

3) Processi di supporto

3.1) Documentazione

3.1.1) Descrizione e Scopo

La documentazione software è l'insieme di informazioni, raccolte testualmente, volte allo scopo di spiegare a quali funzionalità assolve un software, come è strutturato, implementato e come lo si utilizza. Nel contesto del team di sviluppo, la gestione efficace dei processi e delle attività è essenziale per agevolare il lavoro dei membri del team. La tracciabilità e la documentazione dettagliata di ogni fase del progetto sono fondamentali per semplificare le operazioni quotidiane e per facilitare la manutenzione del software nel tempo. Questa pratica non solo contribuisce a garantire una maggiore coerenza e coesione all'interno del team, ma anche a migliorare complessivamente la qualità del risultato finale.
È bene quindi che vengano definite delle regole chiare e concise utili per la stesura di un documento, da seguire durante tutto il ciclo di vita del progetto allo scopo di garantire maggiore comprensione.

3.1.2) Strategia "Documentation as Code"

Durate l'intero svolgimento di progetto abbiamo adottato la strategia `Documentation as Code` che cosnsiste nel gestire la documentazione dell'intero progetto come se fosse il codice sorgente. Questa metodologia di lavoro ha i seguenti aspetti chiave:

- Versionamento;
- Collaborazione;
- Automazione;
- Integrazione continua;
- Distribuzione continua;

Questo approccio favorisce una migliore tracciabilità delle modifiche, piu facilità nella verifica e manutenzione.
I documenti sono scritti in formato testuale secondo le regole di formattazione di Typst, strumento che ci permette di performare gli aspetti appena elencati in maniera semplice e intuitiva, insieme al servizio di controllo di versione distibuito tramite interfaccia web GitHub.

3.1.3) Ciclo di vita dei documenti

Il ciclo di vista di un documento segue i seguenti stati:

- Necessità/Opportunità:
  1. Nasce la necessità per obbligo o per opportunità di un documento;
  2. Pianificazione della stesura del documento;
  3. Assegnazione dei redattori del documento.
- Redazione:
  1. Definizione di una struttura (come segue alla sezione dedicata);
  2. Raccolta di informazioni/contenuti;
  3. Stesura del documento rispettando le norme descritte nella sezione dedicata.
- Revisione/Verifica:
  1. Revisione da parte dei verificatori del corretto utilizzo delle norme;
- Manutenzione:
  1. Avanzamento di versione del documento;

2. Richieste di modifiche derivate da nuove necessità o esigenze. Il documento torna nello stato di redazione.

3.1.4) Strumenti

- Typst: scelta definitiva per la formattazione dei documenti, motivata della comodità con cui è possibile effettuare il versionamento e la revisione dei documenti stessi;
- Overleaf (LaTeX): utilizzato nelle prime fasi del progetto per la realizzazione dei documenti necessari, successivamente sostituito con Typst;
- UML: per la creazione di diagrammi UML, il team ha deciso di utilizzare Diagramas.net.

3.1.5) Grafiche

- Template: i documenti di progetto sono realizzati e caratterizzati da un preciso template, che differisce da quelli utilizzati per verbali e lettere di presentazione. I file da includere per applicare il relativo template sono i seguenti:
  1. big_docs.typ, per la documentazione ufficiale di progetto;
  2. verbals.typ, per i verbali, sia esterni che interni;
  3. p_letters.typ, per le lettere di presentazione.
- Tabelle: le tabelle presentano una classica intestazione del contenuto, i nomi delle colonne in grassetto e nessun altra particolarità, si è scelto di utilizzare una filosofia minimale per non appesantire i documenti. Si applicano colori alternati tra righe adiacenti in modo da facilitarne la lettura, rispettivamente "white" per le dispari, e "luma(230)" per le pari. In alcune occasioni, è possibile se concordato, applicare il colore "c33435" alla riga di intestazione, per motivi puramente estetici.
- Figure: le immagini devono rigorosamente essere accompagnate da relativa didascalia, ed essere ridimensionate in modo ragionevole, senza eccessiva perdità di qualità, contenuto e senza andare ad occupare spazi esagerati rendendo la lettura del documento sgradevole.

3.1.6) Norme tipografiche

- Nome file: I nomi dei file hanno tutti una notazione omogenea tra di loro, ovvero, nomi descrittivi del contenuto, la lettera iniziale è sempre maiuscola e il resto tutto minuscolo. Le parole sono separate da degli underscore e la data viene scritta in formato AAAA-MM-GG senza spazi;
- Sezionamento: Divisione in sezioni X.X.X (con alcune eccezioni) e in caso di ulteriori suddivisioni si utilizzano degli elenchi puntati, la sezione X.X.1 è sempre la descrizione del contenuto di quella sezione. Si cerca sempre di rendere il tutto più semplice possibile per facilitarne la lettura e mantenere ordinato il documento;
- Glossario: Il documento glossario è strutturato similmente ad un dizionario, le sezioni primarie suddividono il documento in ordine alfabetico, la sottosezione contiene la parola, all'interno della sottosezione viene descritta la parola, la sua definizione o l'uso che ne si fa nel progetto;
- Stile del testo: Vengono applicati ai documenti stili di testo per facilita la lettura e per segnalare la funzione di certe parole.
  Parole in blu, carattarezzate anche dal corsivo, segnalano parole inserite all'interno del glossario, *esempio*.
  Parole in grassetto segnalano parole tecniche, **esempio**.

L'enunciazione di acronimi e le spiegazioni di termini tecnici vanno racchiuse fra parentesi tonde, (esempio);

- Elenchi puntati: Gli elenchi puntati vengono creati tramite la tabulazione rispetto alla sezione presente, seguita dal trattino corto "-" o da un "+", spaziati dal nome dell'elemento (rispettivamente rapresentanti un elenco puntato, o un elenco numerato). La definizione dell'elemento segue il nome e separata da i due punti e uno spazio ": ".
  La definizione termina con un punto e virgola ";" così da separare i vari elementi, fatta eccezione per l'ultimo, la cui definizione termina con un punto ".".
  L'uso di quale tipologia di elenco utilizzare (puntato o numerato) è lasciato a propria discrezione.

3.1.7) Struttura documenti

I documenti ufficiali hanno una struttura precisa e comune che deve essere rigorosamente rispettata per i motivi citati nella descrizione.

- Prima pagina, sempre composta dal template esclusivo del team, caratterizzata da:
  - Nome del documento;
  - Versione corrente del documento;
  - Logo dell'università;
  - Logo e nome del gruppo;
  - E-mail del gruppo.;
- Registro modifiche (changelog), informazioni organizzate in una tabella, caratterizzato da:
  - Versione del documento;
  - Data delle modifica;
  - Scrittori che hanno effettuato la modifica;
  - Revisori che hanno revisionato la modifica fatta dagli scrittori;
  - Descrizione della modifica.;
- Indice: ogni documento presenta un indice nella pagina seguente al registro delle modifiche, la strutture è divisa in sezioni X.X.X con il numero della pagina in cui inizia la sezione. La divisione X.X.X presenta i macro-argomenti suddivisi nei loro vari paragrafi a loro volta suddivisi in sezioni più specifiche;
- Contenuto: esposto con la maggiore chiarezza e semplicità, rigorosamente diviso in sezioni secondo i principi di indicizzazione;
- Pié pagina, separato dal contenuto con un'interlinea, caratterizzato da:
  - Logo e Nome del gruppo (sulla sinistra);
  - Numero di pagina (sulla destra) nel seguente formato: pagina [n] / [numero di pagine totali].

Struttura dei Verbali

I verbali differiscono in quanto a struttura rispetto ai documenti di progetto. La pagina iniziale, il registro modifiche, l'indice e pié pagina sono sempre presenti anche in questi documenti.
Vanno rispettate le seguenti sezioni:

- **Verbali esterni**:
  - Durata e partecipanti, semplice elenco di informazioni:
    - Ora;
    - Partecipanti;
    - Partecipante esterno;

  - Mezzo tramite.
- Oggetto, descrizione in breve dei temi trattati durante il meeting;
- Sintesi, descrizione corposa di ciò che si è discusso durante il meeting.

Inoltre i verbali esterni dovranno avere un'ultima sezione dedicata alla firma del partecipante esterno e relativa data così da confermare e validate il documento.

- **Verbali interni**:
  - Durata e partecipanti, semplice elenco di informazioni:
    - Ora;
    - Partecipanti;
    - Mezzo tramite.
  - Temi trattati, descrizione in breve dei temi trattati durante il meeting;
  - Sintesi, descrizione corposa di ciò che si è discusso durante il meeting;
  - Obiettivi fissata, tabella che descrive e traccia, attraverso il numero di issue, gli obiettivi prefissati durante il meeting.

  Inoltre i verbali interni, relativi all'inizio di un nuovo sprint, dovranno possedere una sezione "Nuova distribuzione ruolistica" contenente una tabella (Ruolo, Nome e cognome) riportante i ruoli per il prossimo periodo.

3.1.8) Caratterizzazione dei documenti

- Formali: Sono i documenti che andranno a formare la documentazione software del prodotto. In quanto tali sono sottoposti a versionamento, a processi di verifica e approvazione. Essi comprendono documenti interni, utili quindi ai membri del team di sviluppo, ed esterni, destinati a proponente e committente.

  Complessivamente ne fanno parte:

  - Interni:

    - Norme di progetto, rappresentano il "way of working";
    - Verbali interni a uso consultativo.

  - Esterni:

    - Analisi dei Requisiti;
    - Piano di Progetto;
    - Piano di Qualifica;
    - Glossario;
    - Verbali esterni, attestanti di quanto discusso.

- Informali: Sono i documenti interni non destinati alla divulgazione con esterni e fini a loro stessi. Perciò non necessitano di versionamento. Spesso sono bozze in preparazione a documenti formali, o note e appunti generici.

3.1.9) Contenuti

In questa sezione vengono elencati i contenuti dei vari file di progetto.
Verrà in seguito descritta la parte di introduzione, comune fra i vari documenti, e successivamente una descrione più accurata per ogni documento.

Introduzione
Ogni documento ha come prima sezione quella di introduzione, tale sezione serve per introdurre il documento al lettore e sarà composta dalle seguenti sotto sezioni:

- **Scopo del documento**;
- **Scopo del prodotto**: questa parte spiega lo scopo del nostro prodotto software;
- **Glossario**;
- **Maturità e miglioramti**;
- **Riferimenti**.

Analisi dei Requisiti

Il documento `Analisi dei Requisiti` va strutturato nel seguente modo:

- **Descrizione**: sezione che descrive brevemente il prodotto software, il suo obiettivo, le funzionalità e gli utenti;
- **Casi d'uso**: sezione che identifica e descrive in maniera approfondita i casi d'uso del prodotto.
  Ogni caso d'uso e sottocasi d'uso vanno rappresentati con il relativo diagramma UML, gli attori, le precondizioni, le postcondizioni, lo scenario principale,le estensione e le generalizzazioni (quest'ultime due, se presenti). Nella sezione seguente è possibile consultare nel dettaglio le norme e le modalità con i quali vengono descritti i cari casi d'uso;
- **Requisiti**: sezione che elenca e descrive tutte le richieste e vincoli definiti dal proponente o dedotti dal team di progetto durante un meeting. Ogni requisito è caratterizzato da una descrizione e da un caso d'uso annesso. I requisiti possono essere obbligatori, desiderabili o opzionali, inoltre si fa la distinzione tra requisiti funzionali, di qualità, d'ambiente, di performance e di sicurezza;
- **Elenchi**: questa sezione deve comprendere gli elenchi delle immagini e delle tabelle utilizzate all'interno del documento.

Piano di Progetto

Il documento `Piano di Progetto` va strutturato nel seguento modo:

- **Analisi dei rischi**: sezione nella quale di definiscono e analizzano i potenziali rischi che si possono incontrare e influenzare così il successo del progetto.
  I rischi vanno divisi in rischi personali (RP), rischi organizzativi interni ed esterni (ROI/ ROE), rischi tecnologici/software (RT). Ogni rischio è caratterizzato da una propria descrizione, un grado di rischio (occorenza), la propria pericolosità, da una serie di precauzioni prese a priori, dal rilevamento e da un piano di contingenza;
- **Pianificazione**: sezione che definisce periodi/sprint con le relative attività da svolgere, relative risporse e scadenze da rispoettare. Per ogni sprint vengono definiti i ruili di ogni componente e una stima delle ore rischieste per poter portare a compimento le attività pianificate. In particolare vanno inseriti i metodi di lavoro, la gestione delle comunicazioni, la suddivisione delle attività e distibusione dei ruoli;
- **Preventivo**: sezione che fornisce una stima e ripartizioni delle ore produttive di ogni membro del team andando così a definire la durata necessaria per completare le attività di periodo e il relativo costo. Per semplicità di lettura si utilizzeranno tabelle e grafici;
- **Consultivo**: sezione che analizza la ripartizione oraria effettiva in relazione a quella preventivata con relativo impatto sui costi. Viene anche fatta una analisi retrospettiva andando a definire eventuali cambiamenti nel metodo di lavoro e/o nella pianificazione delle attività da svolgere. Per semplicità di lettura si utilizzeranno tabelle e grafici.

Piano di Qualifica

Il documento `Piano di Qualifica` va strutturato nel seguente modo:

- **Qualità del prodotto**: sezione in cui va strutturata la modilità e le metriche di valutazione del prodotto software, in particolare l'architettura, la documentazione e la qualità del software;
- **Qualità di processo**: sezione in cui va strutturata la modalità e metriche di valutazione del processo, in particolare i processi primari, i processi di supporto e i processi organizzativi;
- **Strategia dei test**: sezione in cui vanno spiegati i test di controllo con relativi grafici e tabelle;
- **Miglioramenti**: sezione in cui vanno indicati i miglioramenti possibili del prodotto e dei processi;
- **Controllo delle metriche**: sezione che funge da *cruscotto* per il controllo delle metriche, in maniera da poter controllare l'avanzamento e la qualità del progetto.

Norme di Progetto
Il documento `Norme di Progetto` va strutturato nel seguento modo:
- **Processi Primari**: sezione in cui vanno descritti i processi primari tra cui fornitura, sviluppo, gestione operativa e manutenzione;
- **Processi di supporto**: sezione in cui vanno descritti i processi di supporto tra cui documentazione, gestione della configurazione, qualifica, revisione e verifica e risoluzione dei problemi;
- **Processi organizzativi**: sezione in cui vanno descritti i processi organizzativi tra cui gestione dei processi, gestione ruolistica e gestione delle comunicazioni, seguite da una descrizone delle infrastrutture, dei processi di miglioramento e da quelli di formazione.

Glossario
Il documento `Glossario` va strutturato nel seguento modo:
Le sezioni suddividono il documento in ordine alfabetico, le sottosezioni avranno come titolo la parola e come descrizione la definizione del termine stesso.

3.2) Gestione della configurazione

3.2.1) Descrizione e Scopo

Il concetto di "gestione della configurazione" abbraccia tutte le pratiche essenziali per gestire lo stato di un prodotto software e di tutti i suoi componenti, compresi i file sorgenti e la documentazione. Questo insieme di norme e procedure non solo fornisce informazioni sullo stato di avanzamento del progetto, ma offre anche un resoconto dettagliato dell'evoluzione nel tempo del prodotto, garantendo che il sistema operi secondo le attese. Un'efficace gestione della configurazione è cruciale per preservare l'integrità e le prestazioni del prodotto software durante il suo avanzamento. Inoltre, dovrebbe facilitare la risoluzione di problematiche e conflitti, assicurando una gestione fluida e efficiente del ciclo di vita del software.

3.2.2) Versionamento

Il versionamento è una procedura fondamentale per la gestione di un progetto. Oltre a tracciare i cambiamenti di ogni *artefatto*, documento o sorgente che sia, permette il rispristino di quest'ultimo ad una sua fase precedente rendendo molto più semplice la gestione di errori e conflitti.
Il un numero di versione è così composto:

vX.Y.Z

dove :

- X rappesenta fasi del documento che suddividono e raccolgono i cambiamenti più significativi apportati all'artefatto anche detti "major";
- Y rappresenta modifiche minori come ad esempio la realizzazione di una sezione o feature le quali si pensa non siano sufficienti a stabilire una nuova "fase" del documento. Sono anche identificati con l'appellativo "minor";
- Z rappresenta piccoli aggiustamenti (anche identificati come "fixes") o migliorie generali.

Ad ogni versione corrispode uno stato del documento revisionato.

Lato documentazione
Il changelog o "registro delle modifiche", strettamente collegato al concetto di versionamento, espone al lettore, il ciclo di vita dell'artefatto, le modifiche effettuate, le problematiche sorte e infine anche la distribuzione dei lavori tra i componenti del team di sviluppo.
Nella documentazione è possibile aggiornare la versione andando semplicemente ad aggiungere un nuovo record nella sezione di changelog del rispettivo file sorgente.
Qui sotto un esempio:

```
changelog: (

    "0.5.0", "2023-11-21", p.baggio, p.carraro, "Stesura sezione 3.2",
    "0.4.0", "2023-11-20", p.passarella, p.carraro, "Stesura sezione 3.1",

)
```

La prima persona inserita identifica chi ha svolto le modifiche sul documento, mentre la seconda, chi ne ha revisionato il contenuto.
Una volta fatto, la compilazione automatica, attuata grazie ad una github action realizzata ad hoc insieme alle funzionalità di scripting che fornisce typst, andrà a creare effettivamente la tabella del registro delle modifiche con all'interno tutte le informazioni specificate e richieste. Si noti che *p* ("people") è una variabile d'ambiente contenente tutti i nominativi dei componenti del gruppo di lavoro e di ulteriori nominativi utili e ripetuti molteplici volte nel corso del progetto.

Lato software
Lato softwawre è possibile aggiornare la versione andando ad eseguire un commint con uno specifico messaggio.
Qui sotto un esempio:

```
git add .
git commit --allow-empty -m " version-mid "
git push
```

In particolare, se si vuole creare una nuova release e quindi modificare il numero di versione bisogna includere nel proprio messaggio di commit (l'ultimo prima di aver aperto la pull request) l'apposita sintassi:
- "… version-major" andrà a modificare il digit più significativo (es: v0.0.0 -> v.1.0.0);
- "… version-mid" andrà a modificare il digit di mezzo (es: v0.0.0 -> v.0.1.0);
- "… version-minor" andrà a modificare il digit meno significativo (es: v0.0.0 -> v.0.0.1).

Quando si vuole creare una nuova release basta aprire una pull request verso il branch main. Una volta terminati i check della action, la nuova release verrà creata in automatico.

Il versionamento del prodotto è eseguito in modo automatico tramite l'uso di GitHub Action realizzate appositamente. Ogni versione è corredata da una breve descrizione stilata anch'essa

in modo automatico trasformando i messaggi di commit registrati durante lo sviluppo di quella determinata versione, in un vero e proprio registro delle modifiche (contenente ad esempio l'aggiunta di nuove feature, la risoluzione di problematiche e/o migliorie e modifiche più generiche).

3.2.3) Repository Documentazione

Per la gestione della configurazione e versionamento il progetto si poggia sul uso di un repository Github. Qui sotto un link alla documentazione ufficiale:

Github Docs.

Struttura
L'attuale struttura del repository è suddivisa in 3 branch:

- main;
- approval;
- sources.

main:
E' definibile come il branch di presentazione, nel quale sono presenti solo artefatti revisionati e approvati dal responsabile di progetto corrente. Su esso è applicata una "branch protection rule" che non ne permette i push diretti e protegge il ramo.

approval:
Come intuibile dal nome, è il branch che rappresenta il main durante lo sviluppo e che garantisce che ciò che entra nel ramo principale sia completamente revisionato e approvato. I suoi contenuti verrano uniti a quelli del main tramite un processo di merge ad ogni conclusione di uno sprint, o una volta che il responsabile di progetto lo ritenga possibile e necessario. Le pull request da qualsiasi ramo verso quello di presentazione verranno infatti reinderizzate a quest'ultimo, che ne andrà a valutare la qualità, accettando il lavoro svolto o rimandandolo al mittente con direttive sul come migliorarlo.

sources:
E' il branch relativo alla produzione della documentazione, perciò contiene solo file di tipo .typ e non è pensato per una sua supervisione esterna. I file sorgenti verranno compilati e resi disponibili automaticamente nel ramo approval.

Nel branch main è disponibile un README.md che ne descrive la struttura di cartelle per consentirne una migliore navigazione. Inoltre il team ha pensato di predisporre una pagina web di presentazione dei contenuti del repository, utilizzando le risorse che github.io mette a diposizione.

Qui sotto un link al repository, e alla pagina di presentazione:

Repository di progetto.

Pagina di presentazione.

Procedura di redazione/revisione/manutenzione
Come appreso dalla sezione precedente, il branch sources è quello relativo alla produzione della documentazione di progetto. Questo ramo contiene quindi i file Typst (.typ) necessari per la stesura e modifica del documento.
I seguenti passaggi descrivono le operazioni da effettuare per poter effettuare le operazioni menzionate:
Consigliamo l'utilizzo dei seguenti programmi e strumenti:

- Visual Studio Code (Download disponibile qui);
- Typst Preview: estensione di Visual Studio Code (Extension ID: mgt19937.typst-preview).

Aprire il programma Visual Studio Code e creare una nuova cartella di progetto. (Passare allo step successivo in caso abbiate già la cartella e la repositori remota settata in maniera corretta). Aprire un nuovo terminale all'interno di VScode, quindi dal menu superiore selezionare `Terminal` e di nuovo `New Terminal`.
Nel terminale digitare:

```
git clone https://github.com/farmacodeunipd/farmacode.git .
```

Dopo qualche secondo potrete notare che sono stati scaricati i file all'interno del repository di progetto.

Se necessario spostarsi nel branch sources (Passare allo step successivo se già all'interno del branch sources).
Digitare nel terminale i seguenti comandi:

```
git checkout sources
```

Ora siamo all'interno del ramo di produzione documenti.

Nel caso in cui si intenda continuare ad esempio la stesura di un documento è opportuno scaricare eventuali avanzamenti da parte di altri membri del team di progetto.
Sempre dal terminale integrato digitare il seguente comando:

```
git pull
```

Procedere con le operazioni da fare sui documenti. Si consiglia l'utilizzo dell'estensione nominata in precedenza così da avere una preview live del documento.

Una volta concluse le operazioni sui file .typ è necessario pushare il lavoro fatto e renderlo accessibile agli altri membri del team per ulteriori aggiornamenti.
Da terminale digitare:

```
git add .
git commit -m "Descrizione delle modifiche"
git push
```

È possibile che vi siano dei problemi dopo aver eseguito il comando `git push`. In quel caso digitare da terminale:

```
git pull
```

Ora potrebbe essere necessario risolvere eventuali conflitti e al termine rieffettuare le operazione al passaggio precedente.

3.2.4) Docker (Integrazione)

L'integrazione delle varie componenti del prodotto è gestito interamente tramite l'uso di alcuni container docker realizzati ad hoc. L'utilizzo di docker facilità inoltre il rilascio del software, sul repository sono infatti disponibili tutte le versione delle immagini per poter ricreare i container in qualsiasi ambiente. Il progetto si suddivide in 4 container differenti:

- db (container adibito al database);
- python-api (container adibito alle api che espongo l'algotirmo di raccomandazione);
- express (container adibito alle api che permettono la comunicazione tra db e web-app);
- react-app (container adibito all'hosting del client web).

3.2.5) Local testing

Per eseguire i test localmente nelle propria macchina basta seguire le seguenti istruzioni:
Prima di tutto bisogna azionare il container Docker:

```
docker-compose up
```

E' possibile anche reperire l'ultima versione delle immagini dal repository del progetto, per velocizzare così il processo:

```
docker pull ghcr.io/farmacodeunipd/mvp/mvp_db:latest
docker pull ghcr.io/farmacodeunipd/mvp/mvp_python-api:latest
docker pull ghcr.io/farmacodeunipd/mvp/mvp_react-app:latest
docker pull ghcr.io/farmacodeunipd/mvp/mvp_express:latest
```

Una volta che il container e tutte le sue immagini hanno concluso il loro avvio, sarà possibili testare i vari servizi con i rispettivi comandi:

- Python-api (Algoritmo e relative API):

  ```
  docker-compose up
  ```
- React-app (Client web):

  ```
  docker exec mvp-react-app-1 npm test
  ```
- Express (API per la connesione tra db e client):

  ```
  docker exec mvp-express-1 npm test
  ```

3.3) Qualifica

3.3.1) Descrizione e Scopo

La qualità di progetto è l'iniseme delle attività e di accorgimenti eseguiti durante il suo sviluppo per garantire che le performance del progetto siano congruenti con gli obiettivi e i requisiti stabiliti. Questo implica l'attuazione di misure e controlli volti a garantire che gli output del progetto soddisfino gli standard di qualità prefissati e rispettino le aspettative stabilite nel contesto del progetto stesso. La gestione della qualità è un elemento chiave per assicurare il successo complessivo del progetto e la soddisfazione delle parti interessate.
Le attività messe in atto per garantirne la qualità sono:

- Rispettare e comprendere le necessità del proponente, sia in termini di quantità che di qualità, in modo da rilasciare un prodotto il più possibile fedele all'idea del cliente;
- Seguire il piano di qualifica per lo sviluppo del progetto, in modo da rientrare nei termini pattuiti, di tempo e di costo.

3.3.2) Testing

La verifica è un processo fondamentale per la valutazione di un prodotto software. Questa attività si svolge nel corso della fase di sviluppo con l'obiettivo di individuare difetti e guasti fin dalla fase iniziale del ciclo di vita del prodotto, garantendo simultaneamente il massimo rispetto dei requisiti imposti dal cliente.
Essa rappresenta un elemento cruciale per garantire la qualità e la conformità del software alle specifiche stabilite. Attraverso metodologie di testing, analisi del codice e altre tecniche di valutazione, si mira a identificare e correggere tempestivamente eventuali anomalie. Ciò non solo contribuisce a prevenire la diffusione di difetti nelle fasi successive, ma assicura anche che il prodotto finale soddisfi le aspettative del cliente.

Analisi statica

La verifica statica, così chiamata poichè non richiede l'esecuzione del prodotto, consiste nell'individuazione e correzione di eventuali problematiche che riguardano convenzioni o metriche stabilite. L'analisi statica riguarda sia il codice del software che la stesura dei documenti allegati. Essa può essere effettuata manualmente, o grazie all'utilizzo di strumenti per l'automazione. Esistono inoltre due modalità differenti per la verifica tramite analisi statica:

- Walkthrought: viene utilizzato nel momento in cui non si sappia dove viene riscontrata la problematica e consiste in una lettura più ampia scorrendo nella sua interezza il documento/codice per trovare l'errore. Questo metodo è sicuramente molto efficace ma anche molto dispendioso in termini di risorse;
- Inspection: A differenza del Walkthrought, questo metodo viene utilizzato quando si è a conoscenza di dove potrebbe essere la problematica. E' quindi un approccio più mirato per l'eliminazione dell'errore e molto meno dispendioso.

Conscio di quanto appena descritto, il gruppo ha deciso di ripiegare sul utilizzo di automazioni qual'ora possibile, in modo da velocizzare e abbattere i costi (almeno nel lungo termine) della pratica.

- Documentazione: Per quanto riguarda la sintassi e correttezza dei file .typ, il gruppo si affida agli strumenti di correzione automatica integrati nel editor predisposto per la stesura dei documenti. Per quanto riguarda invece i controlli sulla correttezza della grammatica e ortografia della lingua italiana, dopo aver testato diversi strumenti di "spell checking" integrabili nel editor, il team ha deciso di attenersi ad un controllo manuale, sicuramente più dispendioso, ma anche più affidabile ed efficace.
- Codice: A proposito del codice, il team ha deciso di utilizzare strumenti automatici di analisi statica integrati tramite action. Prima che il codice possa raggiungere il repository, esso viene infatti sottoposto a vari controlli, prettamente di tipo statico. Il gruppo ha deciso, dopo il consueto studio della tecnologia in esame, di utilizzare ruff, un *linter* per codice Python, munito di varie funzioni e strumenti diversi. Principalmente vengono svolti controlli sull'indentazione e correttezza della sintassi del codice.

Per quanto rigurda l'analisi statica del codice sono stati integrati strumenti automatici quali:

- Ruff (per il codice realizzato in python);
- ESLint (per il codice realizzato in js e react)

Analisi dinamica

L'analisi dinamica è un tipo di verifica che viene fatta grazie all'esecuzione del prodotto, atto a identificare errori e controllare il corretto funzionamento. Questo processo dinamico è complementare all'analisi statica, che si concentra sulla revisione del codice sorgente senza eseguirlo. Insieme, l'analisi dinamica e statica costituiscono una strategia completa per la verifica e la validazione del software durante il suo sviluppo e oltre.

- Documentazione: Per quanto riguarda la sintassi e correttezza dei file .typ è stata creata una github action ad hoc che oltre a compilare questi ultimi si accerta che i documenti non presentino errori che possano compromettere la loro visione una volta avvenuta la conversione in formato pdf. In caso il *parser* trovi delle incongruenze o errori, stampa nel log delle action di github il problema, in modo da facilitarne la correzione. Per quanto riguarda invece i controlli sulla correttezza della grammatica e ortografia della lingua italiana, dopo aver testato diversi strumenti di "spell checking" integrabili sempre tramite action, il team

ha deciso di attenersi ad un controllo manuale, sicuramente più dispendioso, ma anche più affidabile ed efficace.

- Codice: L'analisi dinamica comporta l'esecuzione di vari casi di test durante il funzionamento del codice. L'obiettivo principale di questi test è verificare che il software esegua correttamente, individuando eventuali divergenze tra i risultati ottenuti e quelli attesi.
  Per garantire l'efficacia di tali test, è imperativo automatizzare il processo e renderlo ripetibile. Questo approccio mira a valutare in modo il più oggettivo possibile il prodotto.
  Nel panorama dell'ingegneria del software, la tecnica principale associata all'analisi dinamica è il testing.
  Per massimizzarne l'efficacia, ogni test deve essere deterministico e replicabile, cioè, dati gli stessi input, deve produrre sempre lo stesso output. Inoltre, ogni test deve presentare parametri ben definiti, inclusi la descrizione degli input e degli output, il comportamento atteso del software e le condizioni di esecuzione del test.
  L'efficacia complessiva dei test dipende dalla qualità del codice scritto e dalla corretta identificazione dei requisiti funzionali e non funzionali del sistema. Sta quindi agli sviluppatori identificare un insieme di casi di prova rappresentativi, consentendo così l'applicazione dei test su un numero limitato e significativo di situazioni.
  Vi sono varie tipologie di test:

  1. Test di unità: I test di unità sono una componente fondamentale del processo di testing software. Essi si focalizzano sulla verifica del corretto funzionamento di unità di codice individuali e indivisibili. Possono essere suddivisi in due categorie principali: test strutturali e test funzionali. I test di unità strutturali si concentrano sulla verifica interna dell'implementazione, mentre i test funzionali si focalizzano sul comportamento esterno e sulle funzionalità offerte dall'unità;

  2. Test di integrazione: I test di integrazione hanno lo scopo di verificare le interazioni tra le diverse unità architetturali. Questi test sono fondamentali per garantire che i vari componenti del sistema si integrino correttamente, identificando eventuali difetti derivanti dall'interazione tra le unità. Questi test possono essere concepiti e realizzati seguendo principalmente tre approcci diversi: *top-down*, *bottom-up* e in modo *incrementale* (o misto).
     Queste tre modalità hanno i loro vantaggi, svantaggi e applicazioni. Il team, dopo una riflessione colletiva, ha indivuato il primo elencato come il più naturale da seguire data la scarsa esperienza nell'ambito;

  3. Test di sistema: I test di sistema sono una fase del processo di testing software che verifica il comportamento del sistema nel suo complesso in accordo con i requisiti individuati. Questi test vengono eseguiti dopo i test di integrazione e possono includere diversi aspetti del sistema, tra cui funzionalità, prestazioni, sicurezza e affidabilità. La loro finalità è accertare che il software soddisfi gli obiettivi e le specifiche del progetto. Il loro obiettivo principale è garantire che il sistema complessivo soddisfi gli standard di qualità e le aspettative dell'utente, fornendo un software stabile, sicuro e pratico;

  4. Test di regressione: Questa tipologia di test viene eseguita per verificare che le modifiche apportate a una parte del codice non abbiano causato regressioni o rotture in altre parti del sistema. In altre parole, l'obiettivo dei test di regressione è assicurarsi che le modifiche al software non abbiano introdotto nuovi difetti o compromesso le funzionalità esistenti;

5. Test di accettazionne: I test di accettazione rappresentano la fase finale del processo di testing software, sono volti a verificarne la conformità rispetto ai requisiti specificati e a ottenere l'approvazione da parte degli stakeholder, inclusi clienti e utenti finali. Questi test sono eseguiti per assicurare che il software soddisfi le aspettative e sia pronto per un possibile rilascio.

Per la realizzazione e integrazione dei test (dinamici) automatici sono stati individuati e scelti questi due servizi:
- Pytest (per la relaizzazione dei test del codice realizzato in python);
- Jest (per la realizzazione dei test del codice realizzato in js e react).

Struttura dei test

Tutti i test andrannò definiti nella seguente modalità:

- Identificativo:

  Ogni test deve essere corredato da un identificativo definito come segue (questo anche per favorirne la tracciabilità):

  T[Tipologia][Id]

  dove: "Tipologia" ne indica il tipo (U –> "Unità", I –> "Integrazione", S …), ed "Id" un semplice codice seriale formato da al massimo due cifre, funge da nome.

- Stato:

  Ad ogni test verrà poi associato uno stato che ne rappresenterà il risultato, esso potrà essere:

  1. ND: quando il test non è ancora stato implementato;
  2. Passato: quando il test riporta esito positivo;
  3. Non passato: quando il test riporta esito negativo.

Affinché ogni test sia ripetibile e fornito con precisione, è essenziale specificare nel apposita documentazione:

- Ambiente e requisiti:

  Definire lo stato iniziale dell'ambiente, includendo dettagli sull'hardware e il software necessari per eseguire il test. Questo può includere informazioni sulla configurazione del sistema, versioni del software e condizioni che possono influenzare l'esecuzione del test;

- Attese e funzionamento:

  Individuare ed esplicitare gli input richiesti e i corrispondenti output attesi. Questo comprende la descrizione dettagliata di cosa ci si aspetta che accada durante l'esecuzione del test e i risultati attesi alla sua conclusione;

- Procedure e valutazione:

  Definire le procedure coinvolte nell'esecuzione del test, fornendo istruzioni dettagliate su come condurre il test. Ciò può includere sequenze di azioni da eseguire e descrizioni degli scenari specifici da testare. Inoltre va indicato come valutare se il test è stato superato con successo o se sono emersi eventuali problemi;

- Tracciamento:

  Indicare che funzionalità mira a coprire il test in esame, specificandone requisiti funzionali rispetto l'utente finale.

3.3.3) Validazione

Lo scopo della validazione è quello di confermare la qualità del prodotto software nella sua interezza, assicurando che i requisiti siano stati implementati correttamente come concordato con il proponente.
Perchè un file venga validato, c'è la necessità che passi i test preposti in base al suo tipo, confermando e attestando la qualità del prodotto.

Code coverage

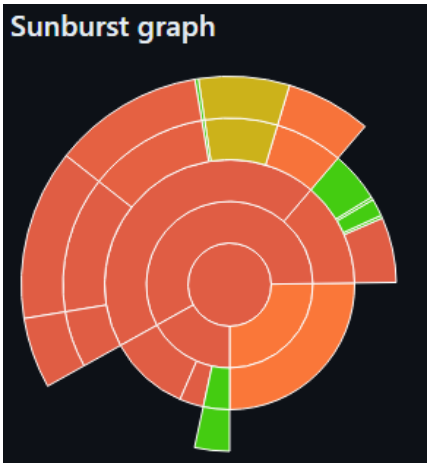

Figura 7: Sunburst graph

Il cerchio centrale rappresenta il progetto nella sua totalità, spostandosi verso l'esterno ci sono le directory, e infine, i singoli file. La grandezza e il colore di ogni "slice" (o spicchio) rappresenta rispettivamente il numero di statements e il coverage.

3.3.4) GitHub Action

linting.yml

```
name: Python and React/JS application
on:
  push:
    branches:
      - develop
    paths:
      - '**.py'
      - '**.js'
      - '**.jsx'
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python 3.11.5
        uses: actions/setup-python@v4
        with:
          python-version: "3.11.5"
      - name: Install dependencies
        run: |
          pip install ruff
          npm install eslint eslint-plugin-react
```

```
        npx eslint . --fix
    - name: Lint with Ruff (Python)
      run: |
        ruff check . --fix
        ruff format .
    - name: Lint with ESLint (React/JS)
      run: |
        npx eslint . --fix
```

- Trigger:
  Il workflow è configurato per essere attivato ogni volta che viene effettuato un push sul ramo "develop" e quando vengono apportate modifiche ai file con estensione .py, .js o .jsx.
- Steps:
  - Checkout del codice: Questo passaggio utilizza l'azione `/checkout@v4` per ottenere una copia del repository;
  - Configurazione di Python 3.11.5: Viene utilizzata l'azione `/setup-python@v4` per configurare l'ambiente Python con la versione 3.11.5;
  - Installazione delle dipendenze: In questo passaggio vengono installati i pacchetti necessari sia per Python che per JavaScript. Per Python, viene utilizzato pip install ruff per installare Ruff, mentre per JavaScript vengono utilizzati npm install eslint eslint-plugin-react per installare ESLint e il plugin per React. Successivamente, viene eseguito npx eslint . –fix per correggere eventuali problemi di stile nel codice JavaScript;
  - Analisi statica con Ruff (Python): Viene eseguita un'analisi statica del codice Python utilizzando Ruff. I comandi ruff check . –fix e ruff format . vengono utilizzati per controllare e formattare il codice Python;
  - Analisi statica con ESLint (React/JS): Infine, viene eseguita un'analisi statica del codice React/JS utilizzando ESLint tramite il comando npx eslint . –fix. Questo passaggio controlla e corregge eventuali problemi di stile nel codice JavaScript e React.
- In sintesi:
  questo workflow si occupa di controllare la qualità del codice Python e JavaScript/React attraverso analisi statiche e correzioni automatiche dei problemi di stile.

test-and-release.yml

```
name: Docker Image Release

on:
  pull_request:
    branches:
      - main

jobs:
  test-and-release:
    runs-on: ubuntu-latest

    steps:
      - name: Set Git user
        run: |
          git config --global user.name "farmacodeunipd"
          git config --global user.email "farmacode.swe.unipd@gmail.com"

      - name: Checkout repository
        uses: actions/checkout@v2
        with:
```

```
          fetch-depth: 0

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Fetch Tags
        run: git fetch --tags

      - name: Get Version
        id: versioning
        run: |
          if [[ $(git tag) ]]; then
            LAST_TAG=$(git tag --sort=-v:refname | head -n1)
            echo "::set-output name=last_tag::$LAST_TAG"
          else
            LAST_TAG="v0.0.0"
            git tag $LAST_TAG
            git push --tags
          fi

          echo "Last Tag: $LAST_TAG"

          IFS='.' read -ra VERSION_PARTS <<< "$LAST_TAG"
          MAJOR="${VERSION_PARTS[0]#v}"
          MID="${VERSION_PARTS[1]}"
          MINOR="${VERSION_PARTS[2]}"

          echo $(git log --format=%B -n 1 $(git rev-list --no-merges -n 1 HEAD))

          if git log --format=%B -n 1 $(git rev-list --no-merges -n 1 HEAD) | grep -q
"version-major"; then
            echo "Version Major Detected"
            ((MAJOR+=1))
            MID=0
            MINOR=0
          elif git log --format=%B -n 1 $(git rev-list --no-merges -n 1 HEAD) | grep -
q "version-mid"; then
            echo "Version Mid Detected"
            ((MID+=1))
            MINOR=0
          elif git log --format=%B -n 1 $(git rev-list --no-merges -n 1 HEAD) | grep -
q "version-minor"; then
            echo "Version Minor Detected"
            ((MINOR+=1))
          fi

          echo "creating new version ..."

          UPDATED_VERSION="v${MAJOR}.${MID}.${MINOR}"
          echo "Updated Version: $UPDATED_VERSION"
          echo "::set-output name=updated_version::$UPDATED_VERSION"

        env:
          GITHUB_SHA: ${{ github.sha }}

      - name: Log in to GitHub Packages
```

```
          run: echo "${{ secrets.PAT }}" | docker login docker.pkg.github.com -u
farmacodeunipd --password-stdin

      - name: Set up Docker Compose
        run: docker-compose up -d

      - name: Run tests and stop if they do not pass
        run: |
          ci_env=$(bash <(curl -s https://codecov.io/env))
         docker exec $ci_env mvp_python-api_1 pytest tests/test_algo.py --cov=. --cov-
report=xml:/tests/coverage.xml --verbose
          docker exec $ci_env mvp_react-app_1 npm test
          docker exec $ci_env mvp_express_1 npm test
        continue-on-error: false

      - name: Copy coverage reports
        run: |
          docker cp mvp_python-api_1:/tests/coverage.xml ./coverage-python-api.xml
           docker cp mvp_react-app_1:/client/coverage/coverage-final.json ./coverage-
react.json
            docker cp mvp_express_1:/express/coverage/coverage-final.json ./coverage-
express.json

      - name: Stop and remove Docker containers
        run: docker-compose down

      - name: Upload Python API coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          file: ./coverage-python-api.xml
          flags: python-api
          name: codecov-python-api

      - name: Upload React application coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          file: ./coverage-react.json
          flags: react-app
          name: codecov-react-app

      - name: Upload Express coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          file: ./coverage-express.json
          flags: express
          name: codecov-express

      - name: Push Docker images
        run: |
          docker images
          IMAGES=("mvp_db" "mvp_python-api" "mvp_react-app" "mvp_express")
          for IMAGE in "${IMAGES[@]}"; do
           # Controllo se l'immagine è stata modificata confrontando l'hash SHA locale
```

```
con quello remoto
            LOCAL_SHA=$(docker inspect --format='{{index .RepoDigests 0}}' $IMAGE)
               REMOTE_SHA=$(docker run --rm docker.pkg.github.com/farmacodeunipd/mvp/
$IMAGE:${{ steps.versioning.outputs.updated_version }} sha256sum /)

            if [ "$LOCAL_SHA" != "$REMOTE_SHA" ]; then
              # L'immagine è stata modificata, la etichetto e la pusho
                   docker tag $IMAGE docker.pkg.github.com/farmacodeunipd/mvp/$IMAGE:
${{ steps.versioning.outputs.updated_version }}
                        docker push docker.pkg.github.com/farmacodeunipd/mvp/$IMAGE:
${{ steps.versioning.outputs.updated_version }}

              docker tag $IMAGE docker.pkg.github.com/farmacodeunipd/mvp/$IMAGE:latest
              docker push docker.pkg.github.com/farmacodeunipd/mvp/$IMAGE:latest
            else
              echo "L'immagine $IMAGE non è stata modificata. Salto il push."
            fi
          done

      - name: Check if release already exists
        id: check_release
        run: |
          VERSION="${{ steps.versioning.outputs.updated_version }}"
          echo "Checking release for version ${VERSION}"
          response=$(curl -s -o /dev/null -I -w "%{http_code}" https://api.github.com/
repos/farmacodeunipd/mvp/releases/tags/${VERSION})
          echo "Response code: $response"
          if [[ $response -eq 200 ]]; then
            echo "Release already exists for version ${VERSION}"
            echo "::set-output name=release_exists::true"
          else
            echo "Release does not exist for version ${VERSION}"
            echo "::set-output name=release_exists::false"
          fi

      - name: Get Commit Messages Since Last Release
        id: commit_messages
        run: |
          LAST_RELEASE_TAG=$(git describe --tags $(git rev-list --tags --max-count=1))
           COMMIT_MESSAGES=$(git log --pretty=format:"- %s" $LAST_RELEASE_TAG..HEAD --
no-merges)
          echo "::set-output name=commit_messages::$COMMIT_MESSAGES"

      - name: Create GitHub Release
        if: steps.check_release.outputs.release_exists != 'true'
        uses: actions/create-release@v1
        with:
          tag_name: ${{ steps.versioning.outputs.updated_version }}
          release_name: Release ${{ steps.versioning.outputs.updated_version }}
          body: |
            Release ${{ steps.versioning.outputs.updated_version }}

            Changes since last release (commit messages):
            ${{ steps.commit_messages.outputs.commit_messages }}
        env:
          GITHUB_TOKEN: ${{ secrets.PAT }}
```

- Trigger:
  Il workflow viene attivato ad ogni pull request aperta verso il ramo principale, main.
- Steps:
  - Imposta l'utente Git: Configura l'utente Git globale utilizzando il nome utente e l'email forniti;
  - Checkout del repository: Esegue il checkout del codice del repository per eseguire le azioni successive;
  - Imposta Docker Buildx: Configura Docker Buildx, un plugin Docker CLI per estendere le capacità di build;
  - Recupera i tag: Recupera i tag dal repository;
  - Ottieni la versione: Determina la versione aggiornata basata sull'ultimo tag e sui commit. Se viene trovata una specifica parola chiave nel messaggio del commit (version-major, version-mid o version-minor), incrementa la parte di versione corrispondente;
  - Accedi a GitHub Packages: Effettua l'accesso al registro Docker di GitHub Packages per caricare le immagini Docker;
  - Imposta Docker Compose: Configura Docker Compose per gestire più container Docker;
  - Esegui i test e interrompi se non passano: Esegue i test per diversi servizi (API Python, app React, Express) all'interno dei container Docker. Il workflow si interrompe se i test non passano;
  - Copia i report di copertura: Copia i report di copertura generati dai test dai container Docker alla macchina locale;
  - Carica i report di copertura su Codecov: Carica i report di copertura su Codecov per ciascun servizio (API Python, app React, Express);
  - Carica le immagini Docker: Etichetta le immagini Docker con la versione aggiornata e le carica nel registro Docker di GitHub Packages;
  - Interrompi e rimuovi i container Docker: Interrompe e rimuove i container Docker creati da Docker Compose;
  - Controlla se il rilascio esiste già: Verifica se esiste già un rilascio per la versione aggiornata su GitHub;
  - Crea la descrizio del rilascio: Reperisce tutti i messaggi di commit avvenuti tra la precedente versione e la creazione di quella corrente, per allegarli come descrizione del rilascio;
  - Crea il rilascio GitHub: Crea un nuovo rilascio su GitHub se non esiste già per la versione aggiornata.
- In sintesi:
  Questo workflow automatizza il versioning, i test e il processo di rilascio delle immagini Docker, garantendo rilasci consistenti e affidabili per il progetto.

coverage-main.yml

```
name: Codecoverage

on:
  push:
    branches:
      - main

jobs:
  codecoverage:
    runs-on: ubuntu-latest
```

```
    steps:
      - name: Set Git user
        run: |
          git config --global user.name "farmacodeunipd"
          git config --global user.email "farmacode.swe.unipd@gmail.com"

      - name: Checkout repository
        uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Set up Docker Compose
        run: docker-compose up -d

      - name: Run tests and stop if they do not pass
        run: |
          ci_env=$(bash <(curl -s https://codecov.io/env))
         docker exec $ci_env mvp_python-api_1 pytest tests/test_algo.py --cov=. --cov-
report=xml:/tests/coverage.xml --verbose
          docker exec $ci_env mvp_react-app_1 npm test
          docker exec $ci_env mvp_express_1 npm test
        continue-on-error: false

      - name: Copy coverage reports
        run: |
          docker cp mvp_python-api_1:/tests/coverage.xml ./coverage-python-api.xml
           docker cp mvp_react-app_1:/client/coverage/coverage-final.json ./coverage-
react.json
            docker cp mvp_express_1:/express/coverage/coverage-final.json ./coverage-
express.json

      - name: Stop and remove Docker containers
        run: docker-compose down

      - name: Upload Python API coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          file: ./coverage-python-api.xml
          flags: python-api
          name: codecov-python-api

      - name: Upload React application coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          file: ./coverage-react.json
          flags: react-app
          name: codecov-react-app

      - name: Upload Express coverage report to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
```

```
token: ${{ secrets.CODECOV_TOKEN }}
file: ./coverage-express.json
flags: express
name: codecov-express
```

- Trigger:
  Il workflow viene attivato ad ogni push sul ramo principale, main.
- Steps:
  - Set Git user: Configura il nome e l'email dell'utente Git per le operazioni successive;
  - Checkout repository: Ottiene una copia del repository;
  - Set up Docker Buildx: Configura Docker Buildx, uno strumento per la compilazione di immagini Docker multi-architettura;
  - Set up Docker Compose: Avvia i container Docker necessari per l'esecuzione dei test;
  - Run tests and stop if they do not pass: Esegue i test per ciascun componente dell'applicazione (Python, React, Express) all'interno dei rispettivi container Docker. Se i test non superano, l'esecuzione del workflow si interrompe;
  - Copy coverage reports: Copia i report di copertura generati all'interno dei container Docker nei file locali del repository;
  - Stop and remove Docker containers: Interrompe e rimuove i container Docker utilizzati per l'esecuzione dei test;
  - Upload Python API coverage report to Codecov: Carica il report di copertura del codice per l'API Python su Codecov, utilizzando il token di accesso fornito come variabile segreta nel repository;
  - Upload React application coverage report to Codecov: Carica il report di copertura del codice per l'applicazione React su Codecov, utilizzando il token di accesso fornito come variabile segreta nel repository;
  - Upload Express coverage report to Codecov: Carica il report di copertura del codice per l'applicazione Express su Codecov, utilizzando il token di accesso fornito come variabile segreta nel repository;
- In sintesi:
  Includere la generazione di report anche per il branch principale (main) è fondamentale poiché fornisce un valore significativo. Ogni volta che viene aperta una pull request, Codecov potrà analizzare e confrontare le coperture del codice tra il branch in sviluppo e il branch principale. Questo fornisce dati preziosi che aiutano a valutare l'impatto delle modifiche proposte e a garantire che il codice integrato mantenga o migliori la copertura del codice già presente nel branch principale. Questo processo contribuisce a mantenere elevati standard di qualità del software e favorisce una migliore comprensione delle modifiche introdotte.

3.4) Revisioni congiunte con il cliente

3.4.1) Descrizione e Scopo

Le revisioni congiunte con il cliente rappresentano un importante aspetto del processo di sviluppo del software, permettendo un coinvolgimento diretto e trasparente con il cliente nell'avanzamento del progetto.
Durante queste sessioni collaborative, gli sviluppatori presentano al cliente gli aggiornamenti, le nuove funzionalità o le modifiche apportate al software. Questo processo consente al proponente di esaminare il prodotto in modo attivo, fornendo feedback tempestivi e suggerimenti oltre ai possibili chiarimenti di natura tecnica o strutturale. Le revisioni congiunte offrono un'opportu-

nità cruciale per allineare le aspettative del cliente con la realizzazione pratica del software, identificare eventuali incongruenze e garantire che il prodotto finale soddisfi appieno le esigenze e le aspettative del cliente.

3.5) Verifiche interne

3.5.1) Descrizione

Le verifiche interne rappresentano un elemento fondamentale all'interno del processo di sviluppo del prodotto, consentendo una stretta collaborazione e trasparenza durante le fasi di avanzamento del progetto.
In queste sessioni collaborative, i componenti del gruppo presentano internamente gli aggiornamenti, le nuove funzionalità o le modifiche apportate al progetto.
Questo approccio consente al team di esaminare attivamente il prodotto, fornendo feedback tempestivi, suggerimenti e affrontando eventuali chiarimenti di natura tecnica o strutturale. Le verifiche interne offrono un'opportunità cruciale per allineare le aspettative del team di sviluppo con la realizzazione pratica del software, identificare eventuali incongruenze e garantire che il prodotto finale soddisfi appieno le esigenze e le aspettative interne precedentemente accordate con il proponente.

3.6) Risoluzione dei problemi

3.6.1) Descrizione e Scopo

Durante il progetto il nostro team ha sempre cercato di operare seguendo un modello di miglioramento continuo, in modo da applicare alle best practices, le soluzioni incontrate fino ad ora alle problematiche più riccorrenti.
La risoluzione dei problemi gioca quindi un ruolo fondamentale per l'avanzamento del progetto didattico in quanto permette di monitorare e risolvere i problemi che possono sorgere durante lo sviluppo o il testing.

3.6.2) Gestione dei problemi

Il nostro gruppo per risolvere problematiche verificate durante lo sviluppo del progetto ha deciso di adottare una strategia sviluppata in quattro punti:
- Registrazione del problema;
- Valutazione del problema;
- Risoluzione del problema;
- Comunicazione e aggiornamento documentazione.

Registrazione del problema
Ogni problema identificato deve essere accuratamente registrato nel sistema di tracciamento dei problemi (ITS). Questo sistema consente di inserire informazioni dettagliate, tra cui:
- Descrizione: breve descrizione, in maniera da far capire il problema che si è riscontarto;
- Stato: indica a che punto si trova la risoluzione del problema: aperto, in lavorazione, pronto per una revisione o chiuso;
- Assegnatario: persona a cui è stata assegnata la risoluzione del problema.
- Data: indica la data in cui è stata aperta la issue;
- Label: indica di cosa si deve occupare la persona preposta, quindi se il problema coinvolge la documentazione o codice.

Valutazione del problema
La valutazione del problema è una fase che precede la risoluzione e prevede un attenta valutazione da parte del gruppo. Questa fase rigurda prettamente problematiche riguardanti il codice la cui discussione avviene attraverso meeting interni. Durante gli incontri si toccano tre temi principali: la causa, l'impatto e il rischio.

Risoluzione del problema
Una volta valutato il problema, sarà la persona incaricata a risolverlo, comunicando tramite il ITS lo stato di avanzamento della risoluzione del problema e sfruttando la strategia valutata precedentemente.

Comunicazione e aggiornamento documentazione
Durante il processo di gestione del problema, è fondamentale mantenere una comunicazione trasparente con il proponente. Questo include la creazione di report periodici sullo stato dei problemi e meeting di aggiornamento.
Deve inoltre, se necessario, essere aggiornata la documentazione in maniera da mantenere una coerenza tra software e documenti.

3.6.3) Gestione dei cambiamenti

La gestione dei cambiamenti è un processo cruciale all'interno dello sviluppo del software, poiché consente di gestire in modo efficace e controllato le modifiche apportate al progetto. Il tener traccia dei cambiamenti è fondamentale in ogni parte del ciclo di vita di un prodotto.
La documentazione dettagliata di ciascun cambiamento, insieme alla sua motivazione, contribuisce a costruire una base solida per la manutenzione futura. La gestione dei cambiamenti viene controllata attraverso il changelog presente in ogni documento, e inoltre riportato nel nome del file la sua versione corrente come spiegato nella sezione del versionamento

4) Processi organizzativi

4.1) Gestione dei processi

4.1.1) Descrizione

In questa sezione vengono trattate tutte le normative utili alla stesura e redazione del documneto "Piano di progetto", gli argomenti che ne fanno parte sono:

- Ruoli e relativa organizzazione: descrizione ed esposizione della strategia utilizzata per la rotazione degli stessi;
- Gestione degli incontri e delle comunicazioni: comprendente della loro suddivisone in base alla tipologia;
- Gestione dell'organizzazione: informazioni relative alle pratiche per la suddivisone delle attività e la loro collocazione temporale nel corso del progetto.

L'obiettivo dei processi organizzativi è quello di arrivare alla creazione del documento denominato "Piano di progetto" nella sua forma il più completa possibile, andando a definire ruoli, attività e la loro collocazione nel tempo. Il documento, oltre ad essere utile al team per gestire l'organizzazione e la gestione dei ruoli di ogni componente, fa da bacheca al committente di quanto appena citato.
Il piano di progetto punta a comprendere tutte le pratiche e metodi riguardati il processo organizzativo e di pianificazione, descrivendone l'applicazione.

4.1.2) Ruoli e relativa organizzazione

La suddivisione in ruoli segue le norme definite nel "Regolamento progetto didattico" sottopostoci dal nostro committente. Qui sotto un breve riepilogo:

Ruolo	**Costo orario**	**Responsabilità**
Responsabile	30	• Coordina l'elaborazione di piani e scadenze; • Approva il rilascio di prodotti parziali o finali (SW, documenti); • Coordina le attività del gruppo.
Amministratore	20	• Assicura l'efficienza di procedure, strumenti e tecnologie a supporto del way of working.
Analista	25	• Svolge le attività di analisi dei requisiti.
Progettista	25	• Svolge le attività di progettazione (design).
Programmatore	15	• Svolge le attività di codifica.
Verificatore	15	• Svolge le attività di verifica.

Tabella 1: Ruoli e responsabilità.

Si noti come i ruoli possano svolgere anche mansioni al di fuori della loro responsabilità in caso di necessità, ovviamente senza venire meno alle pratiche di tracciabilità adottate normalmente dal team.
La loro assegnazione viene gestita dal Responsabile di progetto corrente, il quale confrontandosi con gli altri componenti del gruppo, va a stabilire una rotazione conforme al regolamento. Ogni membro del team dovrà infatti ricoprire ogni carica almeno una volta.
Segue una descrizione più dettagliata per alcuni ruoli, perchè ritenuti più complessi, e rispettive mansioni:

- Responsabile di progetto:

  Il Responsabile si occupa oltre alle attività già sopra elencate, di:
  1. Gestire il repository;
  2. Gestire le risorse umane;
  3. Comunicare con l'esterno, proponente e committenti;
  4. Gestire le comunicazioni e incontri interni;
  5. Gestire imprevisti;
  6. Gestire il bilancio del progetto.

- Amministratore:

  L'Amministratore si occupa oltre alle attività già sopra elencate, di:
  1. Redigere e attuare i piani e le procedure per la gestione della qualità.

- Verificatore:

  Il verificatore si occupa oltre alle attività già sopra elencate, di:
  1. Dare feedback completi e chiari a chi ha prodotto il lavoro successivamente revisionato;
  2. Assicurare che la qualità di quanto prodotto sia conforme agli standard imposti.

4.1.3) Gestione dei "cold start"

Al fine di evitare rallentamenti durante il corso del progetto, dovuti a delle situazioni di "cold start", il team si impegna ad adottare le seguenti pratiche:

- Documentazione dettaglita:

  Ogni membro è tenuto a documentare ogni azione ritenuta non banale e avente valenza e dipendenze future, prima in documenti informali, questo per non rallentare troppo i tempi, successivamente da integrare nella documentazione ufficiale;

- Formazione e Condivisione delle Conoscenze:

  Ogni membro è tenuto, qualora si ritenga necessario, a condividere, oltre che in forma scritta attraverso la documentazione, anche verbalmente le conoscenze e le competenze pregresse o apprese durante gli sviluppi;

- Rotazione Graduale:

  Le rotazioni dei ruoli, quando ritenute necessarie, avverranno in modo graduale. Ciò consentirà a coloro che hanno già sviluppato una certa dimestichezza in un determinato ambito, di supportare chi si avvicina a quel ruolo per la prima volta. In pratica, questa modalità di rotazione riflette l'approccio XP, emulando la pratica del *pair programming*.

4.1.4) Reperibilità dei membri

Ogni membro del gruppo si impegna ad essere reperibile per riunioni sincrone durante la settimana, dal lunedì al giovedì, nel pomeriggio, il venerdì durante la mattinata. In caso di impossibilità di partecipare alle riunioni nelle date stabilite, è obbligatorio informare tempestivamente il Responsabile di progetto. Inoltre, la disponibilità può essere estesa anche durante il weekend in casi di necessità, solitamente preferendo la domenica al sabato.
Durante il corso di uno sprint ogni mebro è libero di gestire le proprie attività di progetto in modo asincrono, a meno che esse non richiedano la collaborazione di più componenti. Ogni membro si assume responsabilmente la gestione di impegni accademici e personali, rispettando le scadenze imposte dal relativo sprint.

Per facilitare l'assegnazione delle attività in relazione agli impegni di ogni componente, il team ha a disposizione un file "Google Fogli" dove sono visualizzabili le proprie disponibilità giornaliere (inserite all'inizio del progetto), dove nel eventualità, è possibile segnare altri impegni inderogabili e sorti in un secondo momento.

4.1.5) Comunicazioni

- **Interne**

  Le comunicazioni interne sono quelle coinvolgenti solo il team, o alcuni dei suoi membri, avvengono tramite mezzi quali:

  – *Telegram*: App di messaggistica istantanea utilizzata ampiamente. È stata creata una chat di gruppo che viene utilizzata per comunicazioni più brevi e di tipo repentino. Solitamente viene utilizzato per stilare un semplice ordine del giorno e per organizzare gli incontri interni del team. Inoltre, sempre trammite Telegram, avvengono comunicazione in privato, che interessano quindi solo alcune parti del gruppo, così da gestire in maniera più regolare ed efficace il flusso di comunicazione interno.

  – *Discord*: Piattaforma VoIP, messaggistica istantanea e distribuzione digitale. È stato creato un server di gruppo suddiviso in vari canali.
  - Canali testuali:
    – General: utilizzato per comunicazioni generali;
    – Link-utility: in esso sono riversati tutti i link utili al gruppo perchè spesso visitati o consultati (ad esempio: link alla tabella condivisa per la gestione costi/ore);
    – Link-brainstorming: racchiude tutti i link a fonti di tipo informativo (ad esempio: link alla documentazione della libreria di python surprise);
    – Domande: è utilizzato per contenere le domande dei vari componenti del team, sia rivolte verso il team stesso, si verso l'esterno (proponente/committenti);
    – Todo-reminder: contiene delle annotazioni su cose da fare nel breve periodo.
  - Canali vocali: sono inoltre presenti molteplici canali vocali per permettere di lavorare in piccoli sottogruppi. In caso di meeting interni ci ritroviamo all'interno di un unico canale così da poter discutere e confrontarci.

- **Esterne**

  Le comunicazioni esterne avvengono tramite i seguenti mezzi:

  – E-mail: Usate per le comunicazioni con proponente e committenti. Principalmente hanno la funzione di concordare meeting, o di esporre quesiti e dubbi;

  – Google Meet: Usato per la comunicazione in videoconferenza con l'azienda proponente. I meeting sono concordati in precedenza tramite e-mail;

  – Zoom: Usato per le comunicazioni in videoconferenza con proponente e committenti. I meeting sono concordati in precedenza tramite e-mail.

4.1.6) Incontri o Meetings:

- **Interni**

Per una migliore gestione degli imprevisti e in generale della pianificazione e organizzazione delle attività, il gruppo ha deciso di adottare due tipologie differenti di incontri interni: "Scheduled Meeting", e "Daily Call". Per questioni di efficenza e praticità si è concordato di adoperare Discord come tramite, la modalità di questi incontri è quindi "da remoto".

- **“Scheduled meeting”**

  Sono i meeting interni che solitamente prevedono la messa a *verbale*. Vengono fissati con cadenza settimanale con data variabile a seconda delle disponibilità dei membri del team, quest’ultima viene regolarmente concordata alla fine del incontro precedente. La loro durata è variabile, e tutte le componenti sono tenute a presenziarvi. Per una migliore gestione del tempo a disposizione è stato deciso di strutturare i meeting come segue:

  – Struttura:

  1. Ordine del giorno: Al inizio di ogni meeting chi ha partecipato alle lezioni del giorno condivide informazioni utili con il team, aggiornadolo su quanto stato discusso con i docenti;
  2. Retrospettiva: Prima di discutere le varie domande e dubbi, il responsabile di progetto corrente stila un breve riassunto di quanto stato fatto nel ultimo sprint, evidenziandone aspetti positivi e negativi. Per facilitare queste operazioni, il responsabile di progetto è tenuto ad arrivare “preparato” al incontro, in modo che sia tutto più scorrevole;
  3. Domande e dubbi: Successivamente vengono discusse domande e dubbi per le quali si ritenga necessaria una discussione collettiva;
  4. Pianificazione prossima: Infine viene effettuata, avendo una migliore idea sul avanzamento del progetto, una pianificazione più mirata per il prossimo sprint, andando a definire effettivamente le attività, ruotando i ruoli e spartendo questi ultimi;
  5. Post meeting: Alla fine di ogni meeting il responsabile di progetto fa un breve resoconto di quanto discusso e lo condivide sul gruppo telegram in modo da aggiornare chi eventualmente non fosse riuscito a presenziare al incontro. Successivamente si adopera anche ad una prima stesura del relativo verbale. Per facilitare queste operazioni ogni meeting interno viene registrato.

- **“Daily Call”**

  Sono incontri di durata mediamente minore, che avvengono giornalmente quando e se ne sorge la necessità. Possono essere richiesti da qualsiasi membro del gruppo, e la partecipazione è richiesta solamente ai sottoinsiemi coinvolti. Solitamente non prevedono la stesura di relativo verbale, ma ciò dipende dagli argomenti discussi e dalla presenza o meno di decisioni importanti. Vengono anche utilizzati per sessioni di lavoro “in pair”.

- **Esterni**

  – Proponente:

  Questi incontri prevedono sempre la stesura di relativo verbale necessitante validazione ed approvazione dal partecipante esterno attraverso la sua firma. Solitamente vengono richiesti dal team. Data e orario, sono concordati a priori durante il meeting precedente, o tramite E-mail, tenendo conto delle disponibilità del proponente e del gruppo. Gli argomenti trattati sono di vario tipo e seguono gli sviluppi del progetto, per facilitarne la discussione vengono esplicitati al proponente tramite mail alcuni giorni prima del incontro. Il mezzo varia a seconda delle necessità del proponente, solitamente vengono utilizzate piattaforme come Zoom e Google Meet.

  – Committenti:

  nd.

4.1.7) Gestione dell'organizzazione: metodologia e pratiche

In modo da migliorare la collaborazione e una migliore gestione del ritmo di avanzamento dei lavori, il team ha preso la decisione di adottare un approccio agile nello sviluppo del progetto, ispirandosi a framework e metodologie ben consolidati come Scrum e XP, ampiamente utilizzati in contesti lavorativi reali.

La filosofia che sottende le strategie di tipo agile è incentrata sull'adozione di pratiche di *Continuous Integration/Continuous Deployment* (CI/CD).

Questa scelta mira a fornire diversi vantaggi e valori aggiunti:

- Favorire il Lavoro di Gruppo:

  Promuove la collaborazione e la comunicazione all'interno del team, incoraggiando la condivisione delle idee e delle competenze;

- Sviluppo Individuale:

  Favorirsce la crescita individuale a livello di conoscenze e competenze, consentendo a ciascun membro di contribuire al massimo delle proprie capacità indipendentemente dal ruolo corrente;

- Miglioramento Continuo:

  Promuove il miglioramento continuo attraverso pratiche di retrospezione, identificando e risolvendo le problematiche in modo tempestivo e continuo;

- Organizzazione Efficace:

  Migliora e facilita l'organizzazione tra i membri del team, assicurando una distribuzione efficace delle responsabilità e delle attività;

- Trasparenza per Proponenti e Committenti:

  Garantisce trasparenza al proponente, consentendo un costante flusso di feedback e una maggiore comprensione del processo di sviluppo. Facilita anche l'analisi da parte del committente per una migliore valutazione del progresso.

L'adozione di pratiche CI/CD si inserisce in questo contesto, contribuendo a garantire una integrazione continua del codice e una distribuzione continua delle nuove funzionalità, riducendo al minimo rischi e migliorando la qualità del software. Questo approccio agile mira a fornire al team una struttura dinamica e flessibile per affrontare le sfide e rispondere alle esigenze del progetto in modo efficiente e tempestivo.

4.1.8) Gestione Milestone e Sprint

Le tempistiche del periodo di progetto sono scandite da milestone e sprint. Il team, sempre rifacendosi ad un approccio di tipo agile, ha definito conseguentemente queste ultime.

- Milestone:

  Rappresentano le revisioni di progetto e gli sprint necessari al loro compimento. Ragionevolmente, ogni sprint verrà a posteriori suddiviso in macro attività pianificandone le fasi, la cui granularità e specificità verranno esplorate durante un periodo più a ridosso dello stesso, avendo un istantanea migliore dello stato del progetto.

- Sprint:

Definiscono finestre di tempo durante le quali il team lavora per portare a termine le relative attività, rispettandone le scadenze.
La durata degli sprint deve essere tassativamente fissata, in modo che ogni componente percepisca l'incobenza delle scadenze e quindi si adegui di conseguenza, evitando così "pigrizie". Questo senza andare a discapito della salute mentale di ogni componente, la quale andrebbe ad incidere sulla qualità di quanto prodotto. Solo in casi eccezionali, come durante gli inizi del periodo relativo al RTB, considerato, di "assestamento", sono ammesse delle variazioni, perlopiù per facilitare l'adeguamento di ogni membro al respettivo ruolo corrente.
Per mantenere un workflow scorrevole, si è concordata la durata di una settimana per sprint. Così facendo ogni membro ha la possibilità di esplorare più ruoli e di gestire il proprio tempo in modo più produttivo.

La loro definizione e collocazione temporale è definita nel documento "Piano di progetto", redatto dal responsabile di progetto.

4.1.9) Gestione di attività e Issue

Per una migliore gestione delle attività di progetto vengono suddivise in due categorie differenti:

- Issues: Rappresentano le prime attività identificate dal team durante la fase di pianificazione generale. Con l'avvicinarsi di un nuovo sprint, queste attività diventeranno più specifiche, passando attraverso un processo di analisi e definizione più approfondito, beneficiando di un cruscotto più chiaro che riflette lo stato di avanzamento del progetto;
- Task: Sono micro attività individuate durante lo sviluppo stesso del corrispettivo sprint.

Per la gestione delle attività relative ai processi primari e di supporto, si utilizza il sitema integrato di *Issues Tracking System* (ITS), di Github. Il ciclo di vita delle azioni segue i seguenti passaggi:

Creazione: L'attività viene definita come un compito da svolgere e viene registrata come una issue su GitHub. Le issue devono essere collegate alla/e board di progetto e al rispettivo sprint utilizzando la milestone allegata. La creazione avviene in maniera semplificata in quanto sono stati creati dei template, tra cui abbiamo:
- Documentazione: template per la stesura della documentazione;
- Revisione: template per la revisione della documentazione;
- Verbale esterno: template per la stesura di un verbale esterno;
- Verbale interno: template per la stesuta di un verbale interno;
- Standard issue template: template per una issue comune.

I template favoriscono la compilazione guidata di tutti i campi della issue in questione. Nella maggior parte dei casi, quindi, le label sono già assegnate. Tuttavia è possibile inserirne molteplici e/o personalizzate in base al caso.
Qui ne segue la lista che ne norma l'uso:
- approval: da utilizzare solo per issue che identificano attività di approvazione, da svolgere da parte del Responsabile corrente. Solitamente una issue finale per sprint;
- bug-fix: denota una issue la cui rispettiva attività mira alla correzione di un bug, o altre problematiche;
- code: rappresentano una nuova feature o integrazione nel codice;
- enhancement: issue riferite ad attività portanti migliorie nel repository;
- documentation: rappresentano issue legate alla stesura di documentazione;
- revision: da utilizzare solo per issue legate alle attività di revisone;

- RTB: deve essere usata solo per issue rappresentanti sprint legati al periodo di RTB;
- PB: deve essere usata solo per issue rappresentanti sprint legati al periodo di PB.

Infine le issue devono avere un nome significativo e possedere una descrizione definita dai template descritti in precedenza:
- Documentazione:
  - Title:

    ```
    Continuazione stesura del file ""
    ```
  - Description:

    ```
    **Descrizione:**

    **Data apertura:**
    ```
- Revisione:
  - Title:

    ```
    Revisione del file ""
    ```
  - Description:

    ```
    **Descrizione:**

    **Data apertura:**
    ```
- Verbale esterno:
  - Title:

    ```
    Stesura del file "Verbale esterno[data]"
    ```
  - Description:

    ```
    **Descrizione:**

    **Data apertura:**
    ```
- Verbale interno:
  - Title:

    ```
    Stesura del file "Verbale interno[data]"
    ```
  - Description:

    ```
    **Descrizione:**

    **Data apertura:**
    ```
- Standard issue template: titolo e descrizione libera.

Assegnazione: Le issue vengono assegnate in modo da rispettare la configurazione ruolistica corrente. Il responsabile si occupa di svolgere questo compito al inizio di un ogni nuovo sprint. Per favorire una gestione più decentralizzata delle responsabilità, ogni componente del team si occuperà di gestire le proprie issue nella board di progetto predisposta all'uso;
Completamento: L'attività viene completata dalla persona incaricata, per poi essere spostata nello stato "ReadyToReview" nella rispettiva board di progetto, in modo da notificarne la revisione. Successivamente verrà chiusa attraverso l'apposita funzionalità di chiusura delle issue in GitHub da chi ne svolge la revisone;
Verifica: Chi ne è incaricato procede a verificare quanto svolto utilizzando strumenti automatici o meno, a seconda di quanto prodotto (Documentazione, codice, …). Nel caso in cui il verificatore non sia soddisfatto del lavoro svolto e vi siano grandi modifiche impossibili da apportare da quest ultimo, informerà l'autore e il Responsabile. L'autore dovrà quindi ritornare su quanto

fatto e apportare le modifiche suggerite dal verificatore. Una volta finito con esito positivo il processo di verifica, il verificatore procederà chiduendo sia la issue relativa all'attività corrisponendete, che la issue relativa alla verifica di tale attività.
Si noti che è stato scelto di avere una issue specifica per la verifica, per rendere ai verificatori più facile organizzare e pianificare il proprio lavoro al inizio di ogni sprint;
Accettazione: Dopo la conferma positiva da parte del Verificatore, a ridosso della fine di ogni sprint, il Responsabile effettua un controllo aggiuntivo, procedendo quindi a chiudere l'issue di approvazione corrispondente e a eseguire il merge nel branch principale di presentazione.
Le singole attività vengono valutate in base alla loro dimensione e alla pianificazione definita, considerando sia il carico di lavoro che le responsabilità associate.
Una più dettagliata descrizione della gestione delle board di progetto è visionabile nel prossimo sotto-paragrafo.

4.1.10) Gestione delle dashboard / Github views

Per una migliore gestione e visualizzazione delle attività di progetto, le issue devono, al momento della loro creazione, essere "linkate" al rispettivo Github Project (RTB, PB). Per ogni Project sono state predisposte due view differenti:

- Kanban:

  Rappresenta una dashboard, suddivisa in 4 colonne differenti:
  1. Todo: qui si trovano le issue ancora non iniziate;
  2. In progress: in questa sezione vi sono tutte le issue il cui svolgimento sta avanzando;
  3. ReadyToReview: questa colonna è adibita alle attività necessitanti una revisione;
  4. Done: qui si trovano tutte le issue completate.

  Al fine di un utilizzo utile della board ogni membro del gruppo è tenuto a gestire le proprie attività di progetto e a tenere sotto controllo la Project view in modo da avere sempre una visione dello stato si avanzamento dello sprint;

- Gantt:

  A differenza di quella precedente, mira a fornire una rappresentazione calendarizzata delle attività di progetto. Ogni issue viene collocata temporalmente in base alla sua data di "presa a carico" (ovvero nel momento in cui il suo stato cambia da todo a in progress) come inizio e alla sua data di completamento (inclusa la revisione) come fine. Questo approccio consente una visione più chiara e strutturata delle tempistiche di ciascuna attività.
  Per una comprensione ancora più approfondita dell'insieme, la board viene suddivisa in milestone, consentendo la visualizzazione singola di ciascuna di esse. Questa suddivisione facilita il monitoraggio e la gestione specifica di ciascun obiettivo intermedio, migliorando ulteriormente la chiarezza e la gestione del progetto nel suo complesso.

4.2) Infrastrutture

L'infrastruttura organizzativa costituisce l'insieme degli strumenti impiegati per facilitare e ottimizzare i processi organizzativi, consentendo uno svolgimento pratico ed efficiente delle attività lavorative. Questa struttura svolge un ruolo cruciale nel fornire il supporto necessario per garantire che le operazioni quotidiane come i processi più complessi vengano gestiti con successo.

Questa infrastruttura può comprendere una vasta gamma di elementi, tra cui software specializzato, piattaforme tecnologiche e strumenti di comunicazione.

4.2.1) Strumenti di supporto ai processi

- GitHub

  E' un servizio di hosting per progetti software, implementazione dello strumento di controllo versione distribuito Git. Viene utilizzato per gestire tutti gli artefatti del progetto.

- Telegram

  Telegram è un servizio di messaggistica istantanea e broadcasting basato su cloud. Viene utilizzato per le comunicazioni interne fra membri del team.

- Discord

  E' una piattaforma di VoIP, messaggistica istantanea e distribuzione digitale. Viene utilizzata per incontri interni fra membri del team

- Google Drive, e suo ecosistema: Fogli e Documenti

  Google Drive è un servizio web, in ambiente cloud computing, di memorizzazione e sincronizzazione online. Viene utilizzato per la gestione di file informali, che variano tra note e appunti realizzate utilizzando Documenti, e tabelle di vario tipo e uso, realizzate con Fogli.

- Google Meet e Zoom

  Sono applicazioni di teleconferenza. Vengono utilizzate principalmente per incontri esterni.

- Gmail

  E' un servizio di posta elettronica, utilizzato per le comunicazioni con esterni.

4.3) Miglioramento

Durante lo svolgimento delle attività e la successiva elaborazione della documentazione, ci poniamo come obiettivo quello di cercare di seguire costantemente il principio di miglioramento continuo. L'obiettivo principale è quello di identificare in modo proattivo le attività, i ruoli e le opportunità di miglioramento, cercando nuove alternative o soluzioni per affrontare sfide emergenti o passate. Questo approccio contraddistinguerà ogni fase del processo, dalla pianificazione, all'esecuzione compresa anche la documentazione.

In sintesi, la manutenzione migliorativa dei processi è un ciclo iterativo che ci consente di adattarci dinamicamente alle esigenze mutevoli del progetto, garantendo una crescita continua e una maggiore efficienza nelle nostre attività.

4.4) Formazione

Per promuovere un miglioramento continuo nelle attività svolte e garantirne il corretto mantenimento, è essenziale che ogni membro del team attraversi un periodo di formazione in autonomia, che si protragga durante tutto il corso del progetto in modo asincrono. Questo periodo di formazione mira a fornire a ciascun membro le competenze necessarie per comprendere e utilizzare in modo efficace le tecnologie e le metodologie operative coinvolte nel progetto.

4.4.1) Complementi all'auto-formazione

- GitHub:
  https://docs.github.com/en;
- Git:
  https://git-scm.com/doc;
- Framework Scrum: https://www.atlassian.com/it/agile/scrum;

- Approccio XP: https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/extreme-programming;
- Documentazione Typst: https://typst.app/docs/;

5) Metriche per la qualità

5.1) Metriche per la qualità di processo

5.1.1) Fornitura:

-MPC1 - Estimated at Completion: indica quanto si prevede che costerà il progetto nel suo complesso, considerando l'andamento attuale e le prestazioni passate del progetto.

Formula:

$$= \frac{\text{BAC} - \text{EV}}{\text{CPI}} + \text{AC}$$

dove BAC (Budget at Completion) è il costo totale preventivato del progetto.

-MPC2 - Estimate to Complete: indica quanto si prevede che sarà necessario spendere per portare a termine le attività rimanenti e completare con successo il progetto.

Formula:

$$= \text{EAC} - \text{AC}$$

- MPC3 - Earned Value: riflette il valore finanziario delle attività che sono state completate con successo fino a un certo punto nel tempo.

  Formula:

  $$= \%\text{dicompletamento} * \text{BAC}$$

  dove "%dicompletamento" rappresenta la percentuale di avanzamento del lavoro effettivo.

- MPC5 - Actual Cost: riflette la somma totale di denaro effettivamente speso per eseguire le attività del progetto fino a un punto specifico nel tempo.

- MPC6 - Cost Variance: indica se il progetto è al di sopra o al di sotto del budget pianificato per il lavoro effettivamente completato.

  Formula:

  $$= \text{EV} - \text{AC}$$

  Dove:

  1. Se $\text{CV} > 0$, significa che il valore guadagnato è superiore al costo effettivo, indicando che il progetto sta procedendo sotto il budget pianificato;
  2. Se $\text{CV} < 0$, significa che il costo effettivo è superiore al valore guadagnato, indicando che il progetto sta superando il budget pianificato;
  3. Se $\text{CV} = 0$, significa che il progetto sta rispettando esattamente il budget pianificato fino a quel momento.

- MPC7 - Planned Value: appresenta il valore pianificato delle attività da svolgere fino a un dato punto nel tempo.

  Formula:

  $$= \%\text{dicompletamento} * \text{BAC}$$

dove a differenza di quanto detto per EV, %dicompletamento rappresenta la percentuale di avanzamento del lavoro pianificata.

- MPC8 - Scheduled Variance: indica se il progetto è in anticipo, in ritardo o in linea rispetto alla pianificazione temporale.

  Formula:

  $$= \text{EV} - \text{PV}$$

  Dove:

  1. Se $\text{SV} > 0$, significa che il valore guadagnato è superiore al valore pianificato, indicando che il progetto è in anticipo rispetto alla pianificazione temporale;
  2. Se $\text{SV} < 0$, significa che il valore guadagnato è inferiore al valore pianificato, indicando che il progetto è in ritardo rispetto alla pianificazione temporale;
  3. Se $\text{SV} = 0$, significa che il progetto è in linea con la pianificazione temporale fino a quel momento.

- MPC8 - Cost Performance Index: è utile per valutare l'efficienza finanziaria di un progetto fino a un determinato momento.

  Formula:

  $$= \frac{\text{EV}}{\text{AC}}$$

  Dove:

  1. Se $\text{CPI} > 1$: Indica che il valore guadagnato è superiore al costo effettivo, indicando un'efficienza finanziaria positiva. Più il CPI è alto, più efficientemente il progetto sta utilizzando i suoi budget finanziari;
  2. Se $\text{CPI} < 1$: Indica che il costo effettivo è superiore al valore guadagnato, indicando un'efficienza finanziaria negativa. Un CPI inferiore a 1 suggerisce che il progetto sta spendendo più del previsto per il valore ottenuto;
  3. Se $\text{CPI} = 1$: Indica che il progetto sta spendendo esattamente ciò che è stato pianificato per ottenere il valore guadagnato.

5.1.2) Sviluppo:

- MPC9 - Requirements Stability Index: indice progettato per misurare la stabilità dei requisiti di un progetto durante il suo ciclo di vita.

  Formula:

  $$= \frac{\text{Numero di requisiti invariati}}{\text{Numero totale di requisiti}}$$

- MPC10 - Satisfied Obligatory Requirements: indice che misura il numero requisiti obbligatori soddisfatti.

5.1.3) Verifica:

- MPC11 - Code Covarage: espressa come una percentuale fornisce un'indicazione della quantità di codice che è stata esaminata e verificata rispetto al totale del codice sorgente.

Formula:

$$= \left(\frac{\text{Linee di codice eseguite/testate}}{\text{Totale linee di codice}} \right) * 100$$

- MPC12 - Passed Test: espressa come una percentuale fornisce un'indicazione della quantità di test passati in seguito a verifica.

5.1.4) Accertamento della qualità:

- MPC13 - Quality Metrics Satisfied: espressa come una percentuale fornisce un'indicazione della quantità metriche soddisfatte in seguito a verifica.

  Formula:

$$\text{QMS} = \frac{\text{NQMS}}{\text{TQM}} * 100$$

  dove: NQMS (Number of Quality Metrics Satisfied) è il numero di metriche di qualità soddisfatte mentre TMQ (Total number of Quality Metrics) è il numero di metriche di qualità totali.

5.1.5) Gestione organizzativa:

- MPC14 - Non-calculated Risk: è il numero di rischi non calcolati presi.

5.2) Metriche per la qualità di prodotto:

5.2.1) Correttezza linguistica:

- MPD1 - Errori ortografici: percentuale di errori ortografici presenti.

5.2.2) Leggibilità:

- MPD2 - Indice di Gulpease: indice di leggibilità di un testo tarato sulla lingua italiana,

viene calcolato attraverso il numero di frasi e lettere ed il risultato è un valore compreso tra 0 e 100, dove 100 indica la massima leggibilità.

5.2.3) Funzionalità:

- MPD3 - Copertura dei requisiti obbligatori: indica la percentuale di requisiti obbligatori soddisfatti;
- MPD4 - Copertura dei requisiti desiderabili: indica la percentuale di requisiti desiderabili soddisfatti;
- MPD5 - Copertura dei requisiti opzionali: indica la percentuale di requisiti opzionali soddisfatti.

5.2.4) Usabilità:

- MPD6 - Facilità di utilizzo: espressa come numero di click, fornisce un'indicazione della complessià di utilizzo del prodotto;
- MPD7 - Tempo per l'apprendimento: espressa in minuti, indica il tempo necessario ad imparare ad utilizzare il prodotto.

5.2.5) Portabilità:

- MPD8 - Versioni browser supportate: indica la percentuale di versioni di browser supportate.

5.2.6) Efficienza:

- MPD9 - Tempo medio di risposta al comando di ricerca: indica il tempo in secondi necessario al completamento di una ricerca dopo aver premuto il comando apposito.

5.2.7) Affidabilità:

- MPD10 - Gestione errori: indica la percentuale di errori che vengono gestiti.

5.2.8) Copertura dei test:

- MPD11 - Branch coverage: indica la percentuale di rami del programma che sono stati eseguiti nella fase di test;
- MPD12 - Statement coverage: indica la percentuale di istruzioni del programma che sono state eseguite nella fase di test;
- MPD13 - Function coverage: indica la percentuale di funzioni che sono state eseguite nella fase di test;
- MPD14 - Line coverage: indica la percentuale di linee di codice che sono state testate.

6) Elenco delle immagini

7) Elenco delle tabelle