2024/25

Documentazione

Versione 1.2

Maffeis Riccardo – Mat. 1085706

Zanotti Matteo – Mat. 1085443

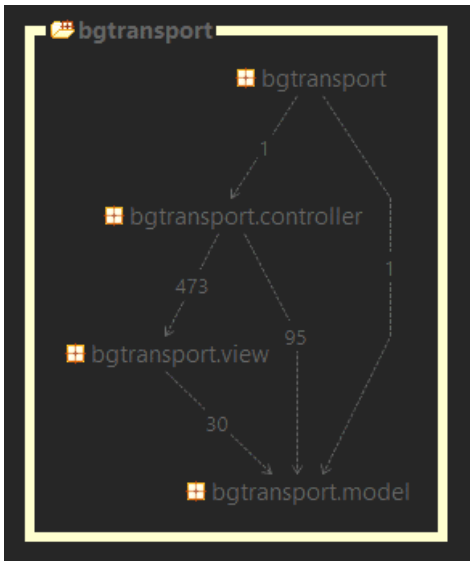
Indice

Software Life Cycle

Abbiamo sfruttato il framework SCRUM per gestire e sviluppare il nostro software.

Inizialmente abbiamo suddiviso lo sviluppo del software in tre pacchetti: bgtransport.controller, bgtransport.model e bgtransport.view. Successivamente abbiamo aggiunto un ulteriore pacchetto, bgtransport, che contiene il Main principale del progetto; gestisce il MainController e il MainModel presenti nei rispettivi pacchetti.

La seguente immagine, recuperata grazie a STAN4J, specifica come sono collegati tra di loro i vari pacchetti all'interno del software:



Inoltre, abbiamo inserito delle tabelle che mostrano come sono stati gestiti gli sprint e i daily:

Sprint	Data	Modalità	Decisioni
0	18/10/2024	Riunione in presenza	Miglioramenti al Project Plan e creazione del file che contiene i requisiti
1	25/10/2024	Riunione in presenza	Miglioramenti al Project Plan, miglioramenti dei requisiti
2	01/11/2024	Riunione in presenza	Termine prima versione ufficiale Project Plan e creazione primi schemi UML
3	08/11/2024	Riunione a distanza	Avanzamento creazione schemi UML e inizio scrittura codice (creazione JOOQ, database e file associati e prime parti della GUI)

4	15/11/2024	Riunione a distanza	Nuove implementazioni nella GUI (mappa, meteo, login) e avanzamento creazione file per i database
5	22/11/2024	Riunione a distanza	Avanzamento creazione GUI (signUp, database, trova la fermata)
6	29/11/2024	Riunione a distanza	Avanzamento della GUI e interconnessione con i database
7	06/12/2024	Riunione a distanza	Miglioramenti alla GUI e inserimento commenti e JavaDoc nel codice
8	13/12/2024	Riunione a distanza	Miglioramenti alla GUI e sistemazione del codice attraverso SonarLint/SonarQube e STAN4J
9	20/12/2024	Riunione a distanza	Miglioramenti alla GUI e sistemazione del codice attraverso SonarLint/SonarQube e STAN4J
10	27/12/2024	Riunione a distanza	Miglioramenti alla GUI e sistemazione del file relativo ai requisiti
11	03/01/2025	Riunione a distanza	
12	10/01/2025	Riunione a distanza	

Sprint	Data inizio	Data fine	Durata
1	21/10/2024	25/10/2024	1 settimana
2	28/10/2024	01/11/2024	1 settimana
3	04/11/2024	08/11/2024	1 settimana
4	11/11/2024	15/11/2024	1 settimana
5	18/11/2024	22/11/2024	1 settimana
6	25/11/2024	29/11/2024	1 settimana
7	02/12/2024	06/12/2024	1 settimana
8	09/12/2024	13/12/2024	1 settimana
9	16/12/2024	20/12/2024	1 settimana

10	23/12/2024	27/12/2024	1 settimana
11	30/12/2024	03/01/2025	1 settimana
12	06/01/2025	10/01/2025	1 settimana

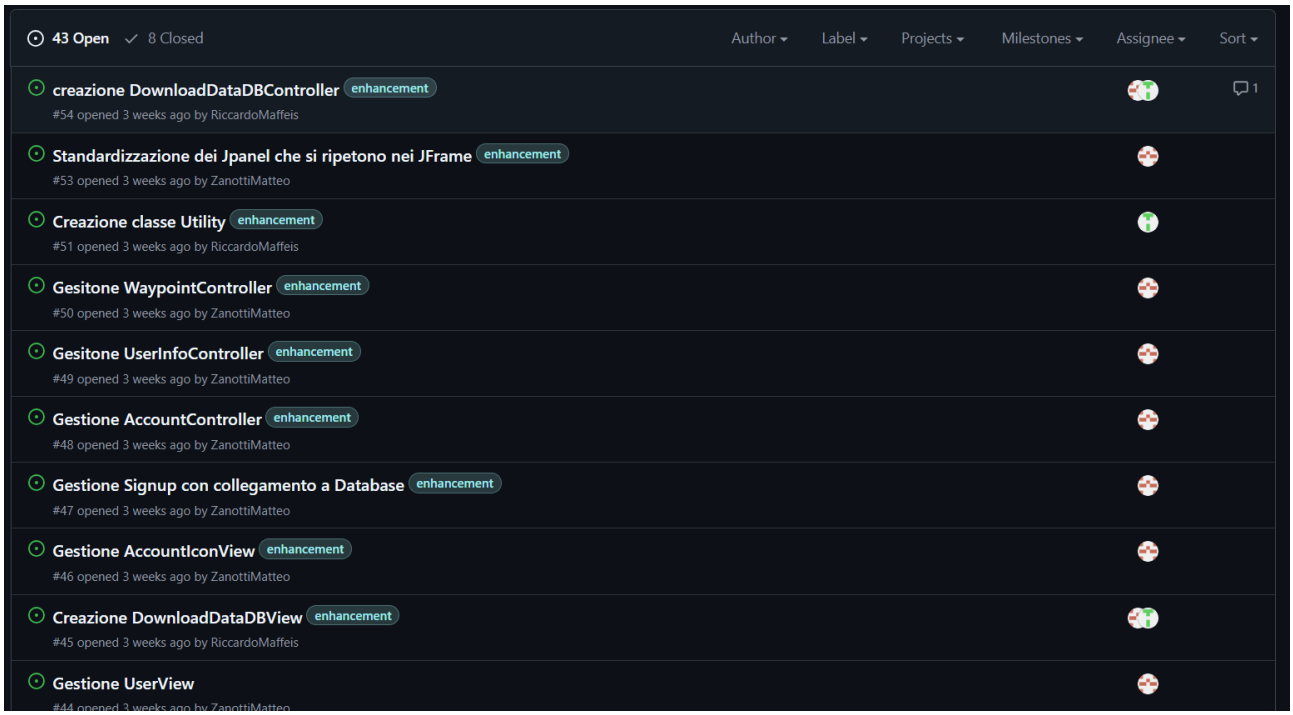
Daily	Data	Modalità	Sprint
1 - 5	21/10 – 25/10 (2024)	Riunioni in presenza giornaliere	1
6-10	28/10 – 1/11 (2024)	Riunioni in presenza e a distanza giornaliere	2
11-15	04/11 – 08/11 (2024)	Riunioni a distanza giornaliere	3
16-20	11/11 – 15/11 (2024)	Riunioni in presenza giornaliere	4
21-25	18/11 – 22/11 (2024)	Riunioni in presenza giornaliere	5
26-30	25/11 – 29/11 (2024)	Riunioni a distanza giornaliere	6
31-35	02/12 – 06/12 (2024)	Riunioni a distanza giornaliere	7
36-40	09/12 – 13/12 (2024)	Riunioni a distanza giornaliere	8
41-45	16/12 – 20/12 (2024)	Riunioni a distanza giornaliere	9
46-50	23/12 – 27/12 (2024)	Riunioni a distanza giornaliere	10
51-55	30/12 – 03/01 (2024/25)	Riunioni a distanza giornaliere	11
56-60	06/01 – 10/01 (2025)	Riunioni a distanza giornaliere	12

Configuration Management

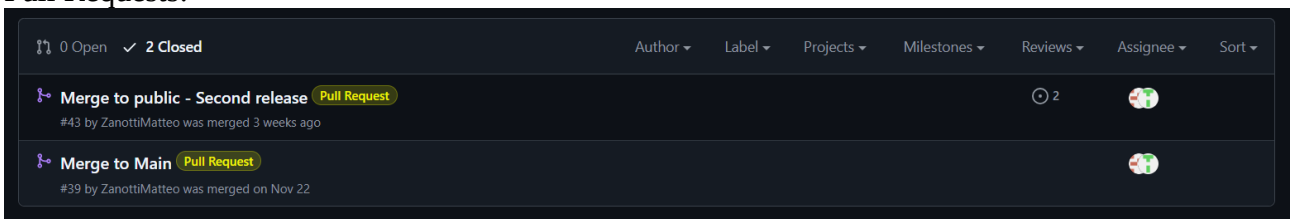
Abbiamo utilizzato GitHub e i relativi comandi (add, commit, push, pull, fetch, clone, merge) e i tools offerti: Project Board, di tipo kanban, con il quale vengono gestiti gli elementi da implementare e i bug da sistemare; RoadMap, la quale possiede una linea del tempo sulla quale vengono distribuiti i vari bug/enhancement in base alla data di creazione; My items, il quale mostra un elenco, in base alle priorità, dei vari bug/ enhancement.

Il link per accedere al repository è il seguente: https://github.com/ZanottiMatteo/BGTransports

Issues:



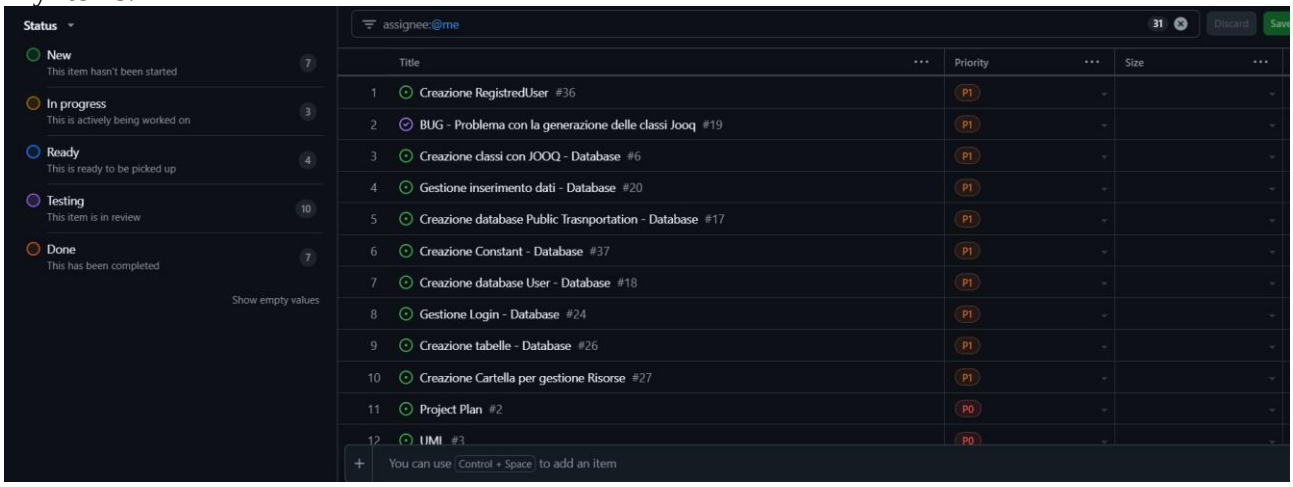
Pull Requests:



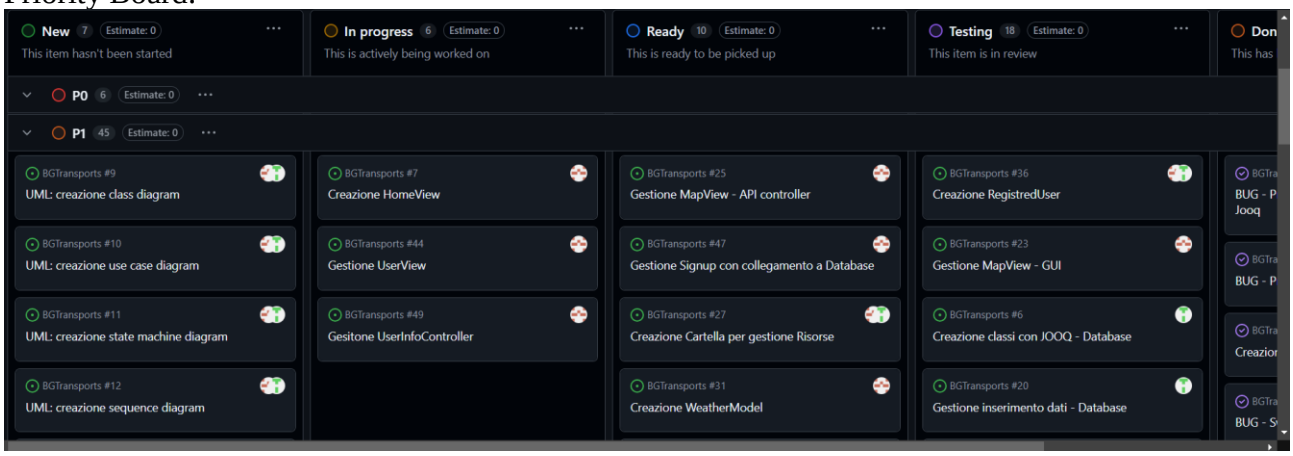
Commit:

Road Map:

My Items:



Priority Board:



People Management and Team Organization

Abbiamo utilizzato un approccio agile per lo sviluppo del software: SCRUM.

Il team lavora basandosi su una buona collaborazione, pianificazione, comunicazione e flessibilità.

Le persone che lavoro al progetto sono 2:

1. Maffeis Riccardo -> Scrum Master che si occupa di far seguire le pratiche Scrum corrette e risolve eventuali problematiche.
2. Zanotti Matteo -> Product Owner che si occupa di definire le funzionalità e le priorità del prodotto

Entrambi ci occupiamo della programmazione del software

Softare Quality

Il software completo deve garantire i seguenti fattori di qualità:

- Correttezza: l'utente deve essere in grado di registrarsi, effettuare il login e vedere tutti i propri dati personali. Successivamente deve essere in grado di interagire con la mappa e ricercare la fermata desiderata.
- Affidabilità: l'utente deve ricevere le risposte alle sue richieste entro poco tempo (2 secondi massimo), senza che si verifichino perdite di dati o che il software si blocchi.
- Efficienza:
- Integrità: ogni utente possiede un proprio ruolo che gli permette di accedere o no alle diverse aree del software, attuando così un controllo sugli accessi al software.

- Usabilità: ogni utente è in grado di comprendere facilmente come muoversi all'interno del software grazie ad un'interfaccia grafica semplice, in parte costituita da pulsanti con icone che specificano la sezione con la quale si vuole interagire.
- Manutenibilità: minimo sforzo per individuare e correggere un errore grazie alla presenza della JavaDoc e dei commenti; inoltre, quasi tutte le stringhe, gli interi, i booleani, che si ripetevano nel codice, sono stati raccolti all'interno di classi e fatti diventare delle costanti.
- Testabilità: minimo sforzo grazie alla presenza di classi di test che permettono di testare le logiche e i database all'interno del software. La parte relativa alla GUI, invece, è testabile direttamente dall'utente, vedendo in prima persona il corretto funzionamento dei menù, pulsanti e mappe.
- Flessibilità: dovrebbe essere semplice aggiungere nuove logiche senza che queste vadano ad intaccare il corretto funzionamento del software, facendo attenzione a integrare le parti grafiche anche con le nuove aggiunte
- Portabilità: minimo sforzo per trasferire il software da un ambiente ad un altro, bisogna solamente seguire le indicazioni presenti all'interno del READMI su GitHub.

Requirement Engineering (capitolo 9)

DOVREBBE indicare come i requisiti sono stati elicitati

DEVE contenere la specifica dei requisiti (ad esempio seguendo lo IEEE 830)

DEVE ???

Documentazione: Documentare i requisiti in documento (2 REQUISITI). La descrizione dei requisiti può seguire lo schema visto in classe dei requisiti presentati in https://github.com/foselab/abz2024_casestudy_MLV/blob/main/Mechanical_Lung_Ventilator%201_5.p df Al suo intero il documento dovrebbe riportare i casi d'uso ed eventuali altri diagrammi UML (ad esempio macchine di stato)

Modelling (Libro UML @ Classroom)

DEVE contenere i diagrammi UML visti a lezione, se non già presentati in altre sezioni della documentazione: - use case diagram (si consiglia di metterlo nella sezione dei requisiti) - class diagram (dal quale si

DEVE generare una prima versione del codice in Java usando Papyrus Designer - si consiglia di metterlo nella sezione software design) - ALMENO UNO state machine diagram (NON DEVE essere troppo semplice, si deve cercare di usare la maggior parte degli elementi che costituiscono uno state diagram come visto a lezione) - ALMENO UN sequence diagram (NON DEVE essere troppo semplice, si deve cercare di usare la maggior parte degli elementi che costituiscono un sequence diagram come visto a lezione). - UN diagramma TRA communication diagram e timing diagram - ALMENO UN activity diagram (NON DEVE essere troppo semplice, si

deve cercare di usare la maggior parte degli elementi che costituiscono un activity diagram come visto a lezione)
- component diagram (si consiglia di metterlo nella sezione in cui si presenta la software architecture)

Software Architecture

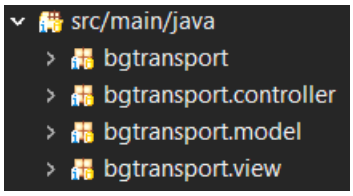
Abbiamo utilizzato uno stile architettonico del tipo Model-View-Controller, con l'aggiunta di un Main principale nel pacchetto "bgtransport":

Model (bgtransport.model): contiene un MainModel che si occupa di effettuare i controlli sui database e molti metodi che si occupano di gestire i database e i dati al loro interno.

View (bgtransport.view): contiene tutta la GUI del codice, la quale viene modifica attraverso il Controller

Controller (bgtransport.controller): si occupa di inizializzare, attraverso il MainController, gestire e modificare la View

Main (bgtransport): è il Main principale attraverso il quale viene fatto partire il software



Di seguito elenchiamo le librerie esterne utilizzate nel progetto:

- **Spring Boot**:
  - spring-boot-starter (versione 3.4.0)
  - spring-boot-starter-web (versione 3.4.0)
- **Logging**:
  - logback-classic (versione 1.5.12)
  - slf4j-simple (versione 2.0.16)
  - log4j-slf4j-impl (versione 2.24.2)
- **Mappe**:
  - jxmapviewer2 (versione 2.8)
  - openlayers (versione 6.1.0) — tramite WebJars
- **Testing**:
  - junit-jupiter-api (scope test)
  - junit-jupiter-params (scope test)
  - h2 (versione 2.3.232, scope test)
- **Database**:
  - sqlite-jdbc (versione 3.43.2.2)
- **ORM/SQL**:
  - jooq (versione 3.19.15)

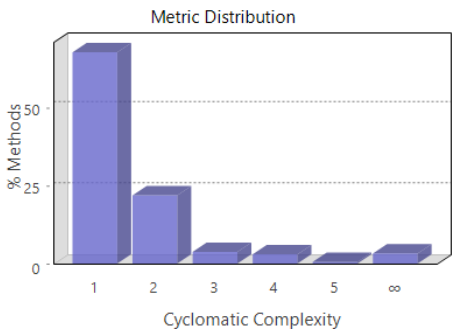
- jooq-meta (versione 3.19.15)
- jooq-codegen (versione 3.19.15)

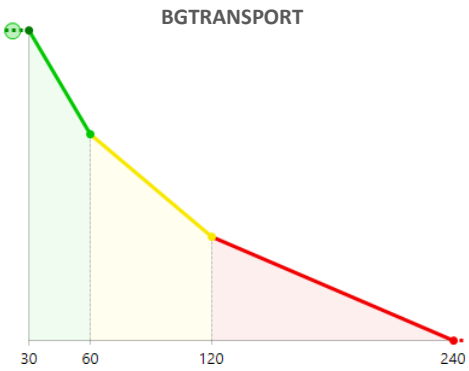
- **Formattazione e Parsing dei Dati**:
  - json (versione 20210307)
  - jackson-databind (versione 2.18.2)
- **UI**:
  - flatlaf (versione 2.6)
- **HTTP Client**:
  - okhttp (versione 4.11.0)

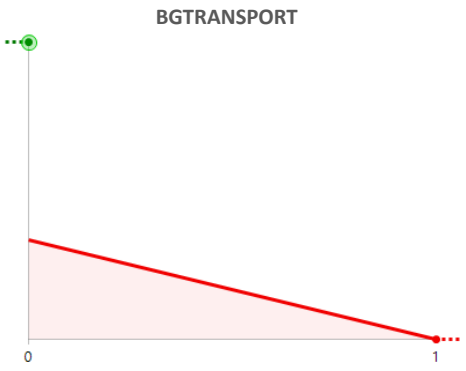
Software Design

Abbiamo analizzato il software con STAN4J e abbiamo trovato i seguenti risultati:

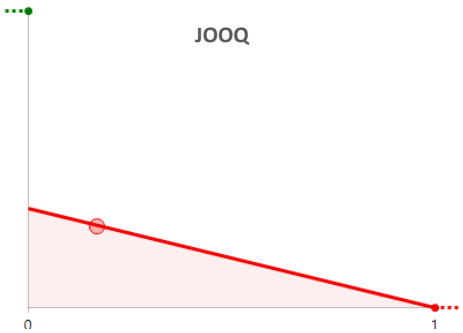
- bgtransport:
  - Cyclomatic Complelxity = 1,71
  - Fat = 5
  - Tangled = 0%







- transportation.jooq.generated:
  - Cyclomatic Complexity = 1,04
  - Fat = 2
  - Tangled = 16,67%
- user.jooq.generated:
  - Cyclomatic Complexity = 1,03
  - Fat = 2
  - Tangled = 16,67%



DEVE contenere una descrizione del design (mediante i diagrammi UML va bene)

DEVE applicare un paio di design pattern visti a lezione

Documentazione: Documentare l'architettura e il design in documento (3 DESIGN).

Software Testing

Il testing del software è stato effettuato dopo gran parte della creazione del codice e si basa su due approcci principali:

1. il primo riguarda il testing delle classi presenti nei pacchetti bgtransport.model e bgtransport.controller attraverso Junit, quindi test che controllano il corretto funzionamento di metodi legati ai database (sono stati testati gran parte dei metodi e delle classi; stiamo parlando di una percentuale di test che si aggira tra il 60% e il 100% del codice presente per ogni classe testata)
2. Il secondo riguarda invece il test della parte grafica e quindi di tutti le classi presenti nel pacchetto bgtransport.view: sono stati testati tutti i pulsanti e le sezioni ad essi collegati, controllando che ogni pulsante aprisse la sezione corretta/svolgesse le operazioni corrette

Software Maintenance

Per quanto riguarda l'attività di manutenzione abbiamo seguito due criteri:

Il primo si basa sulla creazione di classi che contengono la maggior parte delle costanti; questo è stato fatto per non intasare il codice di stringhe e per agevolare gli sviluppatori a effettuare eventuali modifiche nel codice.

Il secondo si basa sul seguire i consigli offerti da SonarLint/SonarQube, andando ad effettuare operazioni di refactoring del seguente tipo:

- modifica del nome delle variabili
- modifica del nome dei metodi
- modifica del nome delle classi
- modifica del nome dei pacchetti
- eliminazioni di parti di codice non utilizzate
- semplificazione di parti di codice troppo complesse

Infine, sono stati aggiunti commenti JavaDoc per permettere una maggior comprensione del codice.

Specifichiamo che queste operazioni sono state fatte durante sia durante l'implementazione del codice, sia dopo la sua conclusione.