



UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Laurea Magistrale in Ingegneria Informatica

Documentazione progetto per il corso di PROGETTAZIONE, ALGORITMI E COMPUTABILITÀ

Prof.ssa:

Patrizia Scandurra

Studenti:

Giorgia BRESSANELLI

Matricola n. 1053903

Lorenzo FERRARI

Matricola n. 1053161

ANNO ACCADEMICO 2023 / 2024

Indice

2	Iterazione 2	3
2.1	Introduzione	3
2.2	UC12: Login	5
2.3	UC21: Logout	5
2.4	UC13: Compilazione modulo emergenza	5
2.5	UC14: Invio allarme emergenza	5
2.6	UC15: Comunicazione con capo servizio	6
2.7	UC16: Visualizzazione stato mezzi SOREU	6
2.8	UC18 e UC31: Scelta ospedale di destinazione	6
2.9	UC20: Visualizzazione report fine emergenza	6
2.10	UC28: Memorizzazione dati	6
2.11	UC29: Ricerca AAT più vicina	6
2.12	Analisi statica	7
2.13	Analisi dinamica	8

2 Iterazione 2

2.1 Introduzione

In questa iterazione si procede con l'implementazione dei casi d'uso essenziali al funzionamento del software (priorità alta).

In particolare, i casi d'uso selezionati per lo sviluppo sono quelli relativi all'**operatore 118**, essendo uno dei due attori essenziali alla comunicazione durante un'emergenza.

Per proseguire con l'implementazione inoltre è risultato necessario aggiornare le classi relative all'operatore 118 e al volontario, aggiungendo una variabile che ne specificasse il ruolo (Figura 1).

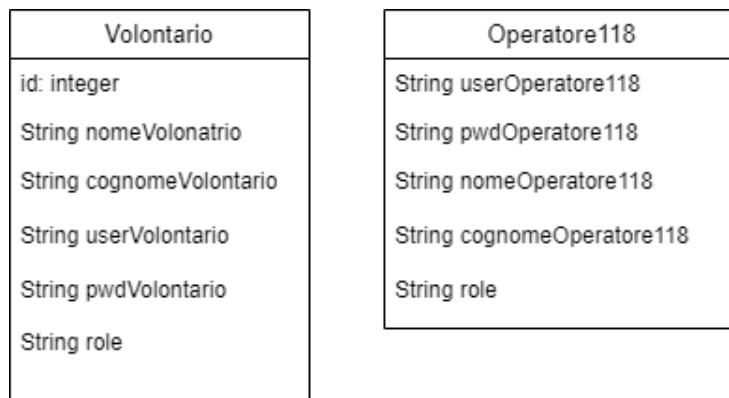


Figura 1: Classi aggiornate

Nello specifico, i casi d'uso scelti sono i seguenti:

- UC12 Login
- UC13 Compilazione modulo emergenza
- UC14 Invio allarme emergenza
- UC15 Comunicazione con capo servizio
- UC16 Visualizzazione stato mezzi SOREU
- UC18 Scelta ospedale di destinazione
- UC20 Visualizzazione report fine emergenza
- UC21 Logout

Inoltre, per poter proseguire successivamente con lo sviluppo dei casi d'uso relativi al capo servizio e per poter integrare i casi d'uso dell'operatore 118, sono stati implementati anche i casi d'uso del sistema:

- UC28 Memorizzazione dati
- UC29 Ricerca AAT più vicina
- UC31 Ricerca ospedali candidati

Di seguito verranno analizzate nel dettaglio le fasi dello sviluppo e i test effettuati.

2.2 UC12: Login

Descrizione Classica fase di login riservata agli operatori 118 e ai capi servizio.

Ogni operatore possiede un proprio username e una password che gli permetteranno di accedere alla schermata di gestione emergenza, differente da quella riservata al volontario. Dopo aver effettuato il login con successo, l'operatore 118 visualizza un'interfaccia che gli permette di

- Rispondere ad una chiamata e contattare una sede AAT da inviare sul posto (azione simulata tramite bottone "*Crea Emergenza*")
- Visualizzare tutte le emergenze concluse
- Visualizzare i report delle emergenze precedenti ed eventualmente generarne altri
- Visualizzare l'emergenza appena inserita dopo aver confermato i dati

2.3 UC21: Logout

Descrizione Classica fase di logout.

2.4 UC13: Compilazione modulo emergenza

Descrizione Dopo aver risposto ad una chiamata al 118, l'operatore visualizza un'interfaccia che gli permette di

- Inserire tramite apposito form i dati ottenuti dalla chiamata
- Visualizzare l'ospedale selezionato dal software ed eventualmente anche gli ospedali candidati per ricevere i pazienti coinvolti nell'emergenza
- Chiudere l'emergenza

2.5 UC14: Invio allarme emergenza

Descrizione Coincide con la chiamata alla sede AAT la quale potrà prendere in carico l'emergenza oppure rifiutarla.

Per semplicità, sono state considerate solamente le squadre aventi lo stato "IN SEDE" come valide per la risposta alle chiamate d'emergenza.

In seguito all'inserimento dei dati ottenuti dalla richiesta di soccorso, l'operatore conferma i dati memorizzati e visualizza lo stato attuale dell'emergenza appena inserita.

L'associazione dell'ultima emergenza con la squadra a cui ne è stata assegnata la gestione,

risulta come "*in attesa*" poiché ancora non sono stati gestiti gli aggiornamenti on-line. In futuro possono essere utilizzati dei Web Socket per fornire questa funzionalità.

2.6 UC15: Comunicazione con capo servizio

Descrizione Questa fase è rappresentata dalle informazioni passate tramite form che vengono caricati per entrambe le parti. Le informazioni sono le medesime citate nell'Iterazione 0.

Per poter sviluppare questo caso d'uso è stato necessario aggiungere uno step intermedio che permetta di memorizzare i dati prima che vengano inviati ai rispettivi form.

2.7 UC16: Visualizzazione stato mezzi SOREU

Descrizione Visualizzazione dell'elenco dei mezzi disponibili per il dipartimento.

2.8 UC18 e UC31: Scelta ospedale di destinazione

Descrizione Visualizzazione degli ospedali disponibili in ordine di vicinanza, calcolata sfruttando le coordinate dell'emergenza e delle strutture stesse. L'operatore ad ora non può forzare la scelta di una struttura che non coincide con la prima suggerita dal software. Si prevede questa implementazione in eventuali sviluppi futuri.

2.9 UC20: Visualizzazione report fine emergenza

Descrizione Visualizzazione delle emergenze memorizzate nel sistema con anche la possibilità di scaricare il documento pdf per ciascuna delle emergenze inserite.

2.10 UC28: Memorizzazione dati

Descrizione Memorizzazione nella base di dati di tutte le informazioni campionate durante la chiamata. Questa fase è essenziale per il momento in cui verrà generato il report di emergenza ma anche per la fase di "comunicazione" tra operatore e capo servizio.

2.11 UC29: Ricerca AAT più vicina

Descrizione Ricerca della sede da contattare più vicina al punto in cui ha avuto luogo l'emergenza. Il calcolo viene eseguito sfruttando la distanza tra i due punti, grazie alla conoscenza delle coordinate di entrambi.

L'operatore 118 può visualizzare direttamente i dati della squadra a cui l'emergenza è stata associata dal programma.

2.12 Analisi statica

Come citato nell'iterazione 0, per l'analisi statica del codice è stato utilizzato il plugin *Qodana*, integrato con *IntelliJ*, software utilizzato per la scrittura del codice.

Qodana permette di verificare l'esistenza di problemi come bugs, dichiarazioni confuse o non utilizzate, violazioni relative a convenzioni adottate per la codifica.

In particolare, in seguito all'analisi effettuata sul codice, sono stati ridefiniti dei metodi mentre altri sono stati completamente rimossi.

Le ulteriori problematiche evidenziate dal tool, visibili in figura 2, sono relative ad un uso non convenzionale di alcune classi per il trasferimento dei dati.

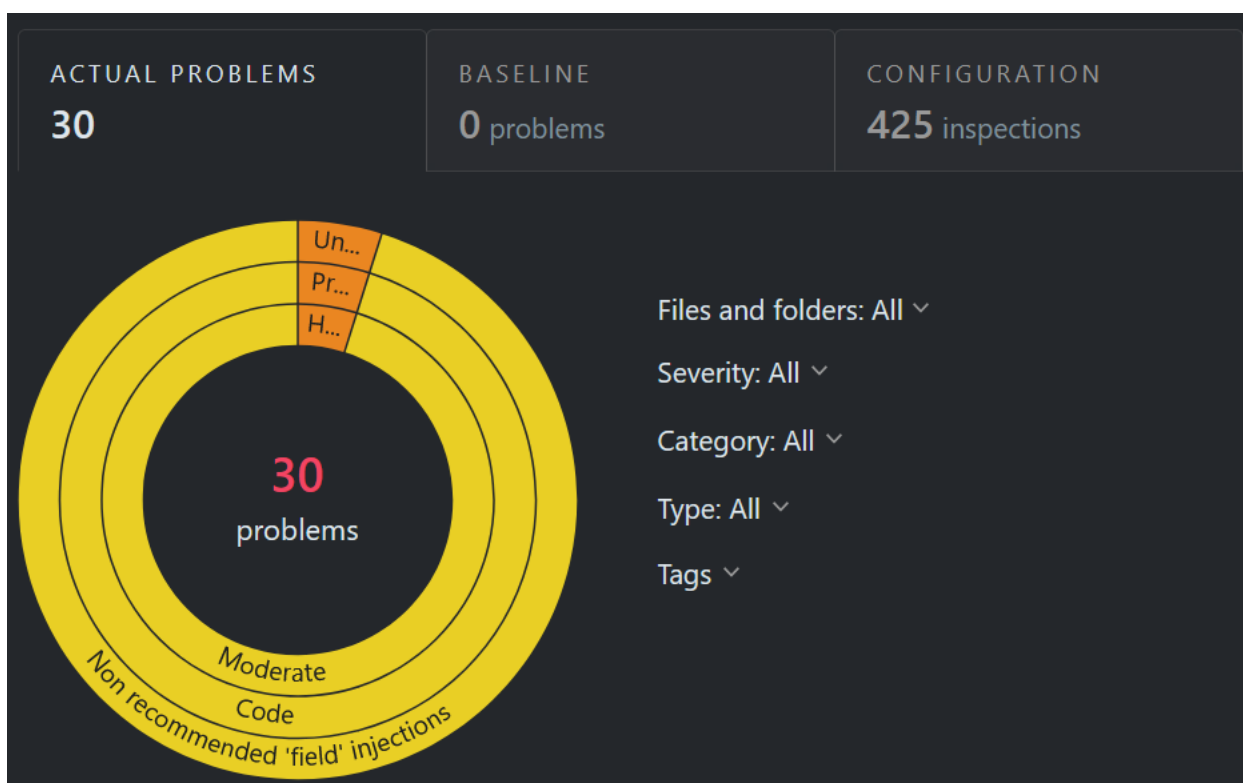


Figura 2: Risultato grafico analisi statica

2.13 Analisi dinamica

Per quanto riguarda la sezione relativa all'operatore sono state testate le seguenti API tramite IntelliJ:

Elenco di tutti i mezzi

GET http://localhost:8080/api/mezzo/all

Content-Type: application/json

X-XSRF-TOKEN: 8de86742-f827-4dd2-8685-759f975b538b

Report dettagliato dell'emergenza numero 1

GET http://localhost:8080/api/emergenza/reportEmergenza/1

Content-Type: application/json

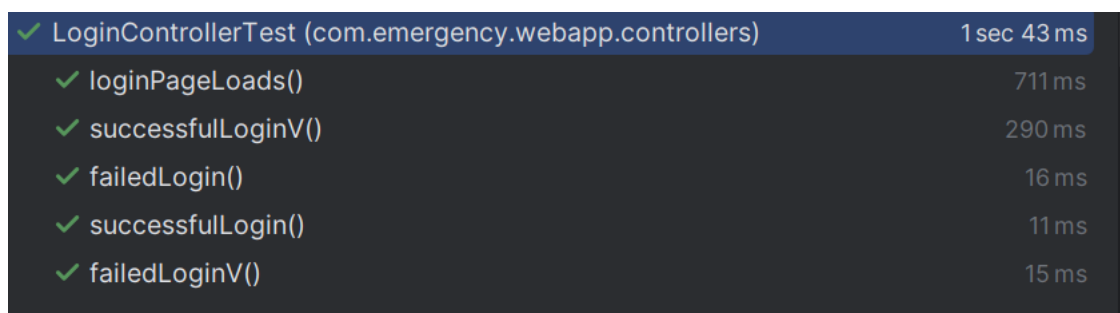
Generazione report dell'emergenza numero 1

GET http://localhost:8080/api/emergenza/generaPDF/1

Content-Type: application/json

Sfruttando *JUnit* i test eseguiti per la fase di login hanno avuto esito positivo, come si può vedere nella figura 3.

In merito ai test per le funzionalità riservate all'operatore 118 si rimanda alla dimostrazione pratica dell'esecuzione dell'applicativo. Non è stato possibile effettuare i test tramite JUnit a causa della presenza del componente SpringBoot Security che limita l'accesso al login.

A screenshot of a JUnit test runner interface showing the results of a test suite. The background is dark with light-colored text. At the top, a summary line shows a green checkmark, the test class name 'LoginControllerTest (com.emergency.webapp.controllers)', and the total execution time '1 sec 43 ms'. Below this, a list of individual test methods is shown, each preceded by a green checkmark and followed by its execution time in milliseconds.

✓ LoginControllerTest (com.emergency.webapp.controllers)	1 sec 43 ms
✓ loginPageLoads()	711 ms
✓ successfulLoginV()	290 ms
✓ failedLogin()	16 ms
✓ successfulLogin()	11 ms
✓ failedLoginV()	15 ms

Figura 3: Esiti esecuzione test login con JUnit

In particolare, i metodi implementati nella classe di test relativa ai login sono:

```
public class LoginControllerTest{
    //declarations
    @Test
    public void loginPageLoads() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/login"))
                .andExpect(MockMvcResultMatchers.status().isOk());
    }

    @Test
    public void successfulLogin() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/login")
                .param("username", "11802")
                .param("password", "albuca"))
                .andExpect(MockMvcResultMatchers.status()
                        .is3xxRedirection())
                .andExpect(MockMvcResultMatchers.redirectedUrl
                        ("/api/emergenza/all"));
    }

    @Test
    public void failedLogin() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.post("/login")
                .param("username", "11802")
                .param("password", "aaaaa"))
                .andExpect(MockMvcResultMatchers.status()
                        .is3xxRedirection())
                .andExpect(MockMvcResultMatchers.redirectedUrl
                        ("/login?error=true"));
    }
}
```