

Deduplicator GUI

Titolo del progetto:	Deduplicator GUI
Alunno/a:	Fadil Smajilbasic
Classe:	SAM I4AC
Anno scolastico:	2019/2020
Docente responsabile:	Geo Petrini

1	Introduzione	4
1.1	Informazioni sul progetto.....	4
1.2	Abstract.....	4
1.3	Scopo.....	4
2	Analisi	5
2.1.1	Analisi del dominio	5
2.1.2	Analisi e specifica dei requisiti	5
2.1.3	Use case	9
3	Pianificazione.....	9
3.1	Analisi dei mezzi	13
3.1.1	Software.....	13
3.1.2	Hardware	13
4	Progettazione.....	14
4.1	Design dell'architettura del sistema	14
4.2	Modifiche da apportare al progetto Deduplicator	14
4.2.1	Modifiche thread di scansione.....	15
4.2.2	Ricavare lo stato della scansione.....	15
4.2.3	Correzione del tempo di esecuzione dello scheduler	15
4.3	Design delle interfacce.....	16
5	Implementazione	22
5.1	Framework Vaadin.....	22
5.1.1	Maven	23
5.2	Correzione errori primo progetto	23
5.2.1	Nuovo metodo di scansione.....	23
5.2.2	Ottenimento stato della scansione	24
5.2.3	Correzione tempo scheduler	25
5.3	Layout generale dell'applicazione	25
5.3.1	MainLayout	26
5.3.2	Client.....	27
5.3.3	LoginView	32
5.3.4	PathView.....	35
5.3.5	ScanView	39
5.3.6	ReportView	40
5.3.7	ScheduleView	44
5.3.8	DashboardView.....	46
5.3.9	Script di installazione	47
5.4	Configurazione	48
5.5	Logger.....	48
5.6	Gestione Autenticazione	50
6	Test.....	52
6.1	Protocollo di test	52
6.2	Risultati test	58
6.3	Mancanze/limitazioni conosciute.....	63
7	Consuntivo.....	63
8	Conclusioni	65

8.1	Sviluppi futuri	65
8.2	Considerazioni personali.....	65
9	Bibliografia	65
9.1	Sitografia.....	65
10	Glossario.....	65
11	Allegati	66

1 Introduzione

1.1 Informazioni sul progetto

Questo progetto è estensione del progetto del primo semestre dell'anno scolastico 2019/2020 dove è stato creato un servizio per eseguire la ricerca e la gestione di duplicati di uno o più percorsi definiti dall'utente. Nel progetto del primo semestre è stata creata una GUI primitiva e incompleta.

Il progetto ha inizio in data 23.01.2020 e finisce il 06.04.2020, il docente responsabile è Geo Petrini.

1.2 Abstract

The project that was terminated in the first semester of the 2019/2020 year consisted in the creation of a service to discover and manage duplicate files in one or more paths defined by the user, it had a primitive and incomplete GUI.

This project has as the primary objective the creation of a modern GUI in order to use and manage all the functionalities of the service. The service must have a modular approach, allowing additions to be implemented in the future. Furthermore, there are some bug fixes to be made to the service.

1.3 Scopo

Con questo progetto si vuole creare una interfaccia grafica, moderna, per l'utente con il scopo di facilitare l'utilizzo e la gestione del servizio. L'implementazione deve avere un approccio modulare per permettere l'aggiunta di ulteriori moduli in futuro come la scansione di tracce audio o immagini.

2 Analisi

2.1.1 Analisi del dominio

Al momento il servizio creato nel primo progetto del semestre ha una GUI primitiva e incompleta, al mercato non esiste nessuna soluzione che potrebbe essere collegarsi e utilizzare le caratteristiche del servizio.

Il prodotto sarà sviluppato in modo da poterlo utilizzare tramite il browser e ogni servizio avrà una sua GUI che lavora in parallelo.

Gli utenti di questo prodotto saranno utenti singoli che useranno il programma per scopo personale ma si può utilizzare anche per lo scopo di piccole aziende con un paio di server.

L'interfaccia utente sarà utilizzata da utenti con conoscenze minime quindi dovrà eseguire tutti i controlli sui dati inseriti dall'utente.

2.1.2 Analisi e specifica dei requisiti

I requisiti per questo progetto sono stati definiti dal docente responsabile Geo Petrini.

L'interfaccia utente dovrà avere un aspetto moderno ispirandosi al material design, un layout responsive e dovrà essere semplice e intuitiva per l'utilizzatore. Tramite l'interfaccia utente si dovranno inoltre poter configurare le impostazioni del servizio, quest'ultima parte è anche da aggiungere al servizio.

Il progetto deve essere strutturato in modo da permettere delle future espansioni di funzionalità.

ID: REQ-01	
Nome	GUI moderna
Priorità	1
Versione	1.0
Note	L'interfaccia utente deve essere moderna e ispirata al material design

ID: REQ-02	
Nome	Layout responsive
Priorità	1
Versione	1.0
Note	L'interfaccia utente si deve adeguare a ogni grandezza di schermo del utente per permettere sempre una visuale utilizzabile.
Sotto requisiti	
01	Visuale desktop
02	Visuale tablet
03	Visuale mobile

ID: REQ-03	
Nome	GUI chiara e semplice
Priorità	1
Versione	1.0
Note	L'interfaccia utente deve essere chiara e semplice

ID: REQ-04	
Nome	Lo stato delle scansioni
Priorità	1
Versione	1.0
Note	Nella GUI deve essere possibile visualizzare lo stato e una stima del tempo.
Sotto requisiti	
01	Modificare la parte del servizio per permettere di ricavare lo stato della scansione
02	Rappresentare lo stato con una barra di caricamento
03	Implementare una stima del tempo necessario al completamento della scansione

ID: REQ-05	
Nome	Pannello per la gestione del servizio
Priorità	1
Versione	1.0
Note	Nella GUI ci deve essere la possibilità di gestire completamente il servizio
Sotto requisiti	
01	Implementare un modo di cambiare la password
02	Implementare un modo per cambiare l'username
03	Implementare la funzionalità di svuotare il file di log
04	Implementare la funzionalità di scaricare il file di log
05	Implementare la funzionalità di cambiare la posizione del file di log

ID: REQ-06	
Nome	Architettura modulare
Priorità	1
Versione	1.0
Note	Il servizio deve essere sviluppato in modo modulare per permettere dei sviluppi e espansioni future.

ID: REQ-07	
Nome	Correzione mancanze primo progetto
Priorità	1
Versione	1.0
Note	Bisogna correggere alcuni errori del progetto del primo semestre
Sotto requisiti	
01	Modificare la scansione come discusso con il docente responsabile
02	Correggere tempo dello scheduler
03	Ricavare stato della scansione

ID: REQ-08	
Nome	Avvio semplice della GUI
Priorità	1
Versione	1.0
Note	La interfaccia utente deve essere semplice da avviare/eseguire.

2.1.3 Use case

Il prodotto avrà diversi casi d'uso, essi sono rappresentati con il seguente diagramma

Prima di fare qualsiasi operazione l'utente deve aver fatto l'accesso al server, questo non è rappresentato nello schema poiché farebbe lo schema molto complicato da leggere.

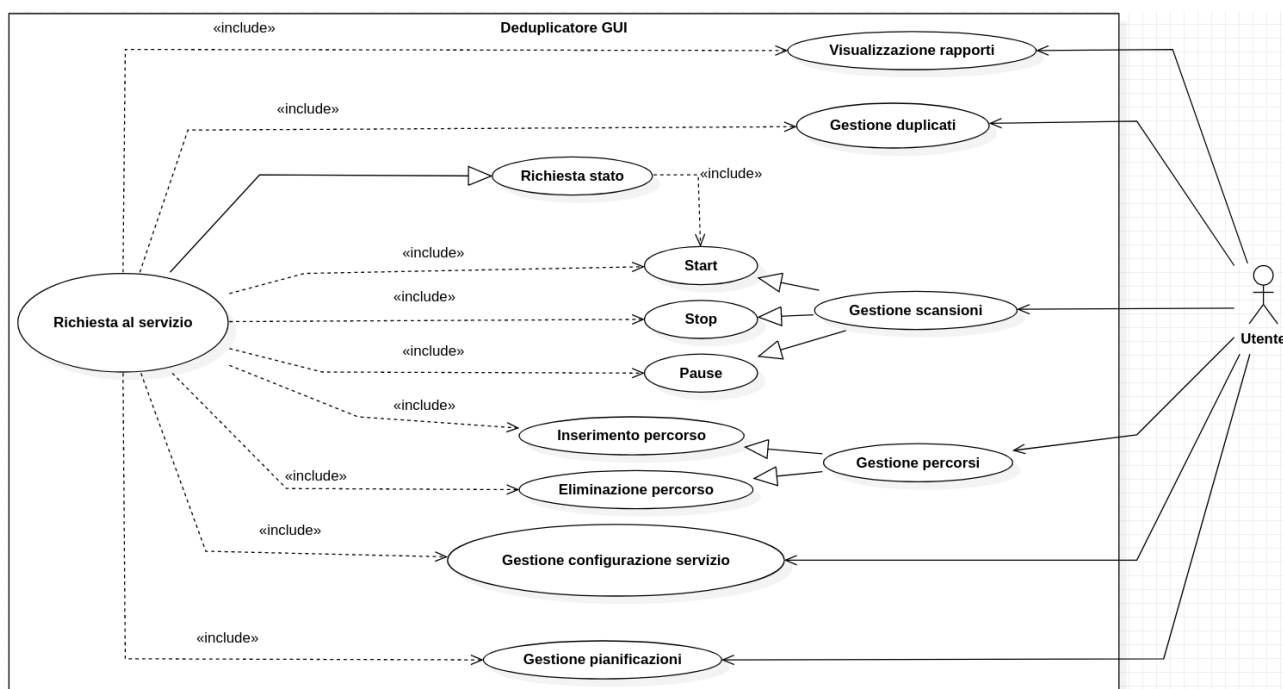


Figura 1 Diagramma use case

3 Pianificazione

Nella figura 2 si può vedere la pianificazione iniziale, la durata del progetto ammonta a circa 118 ore.

Il progetto verrà sviluppato seguendo un modello a cascata con l'eccezione che i test verranno fatti man mano che vado avanti con le parti d'implementazione. Il progetto essendo modulare non avrà bisogno di test che controllino l'integrità dell'intero sistema unico, poiché ogni modulo è indipendente.

Il diagramma di Gantt preventivo è presente tra gli allegati del documento.

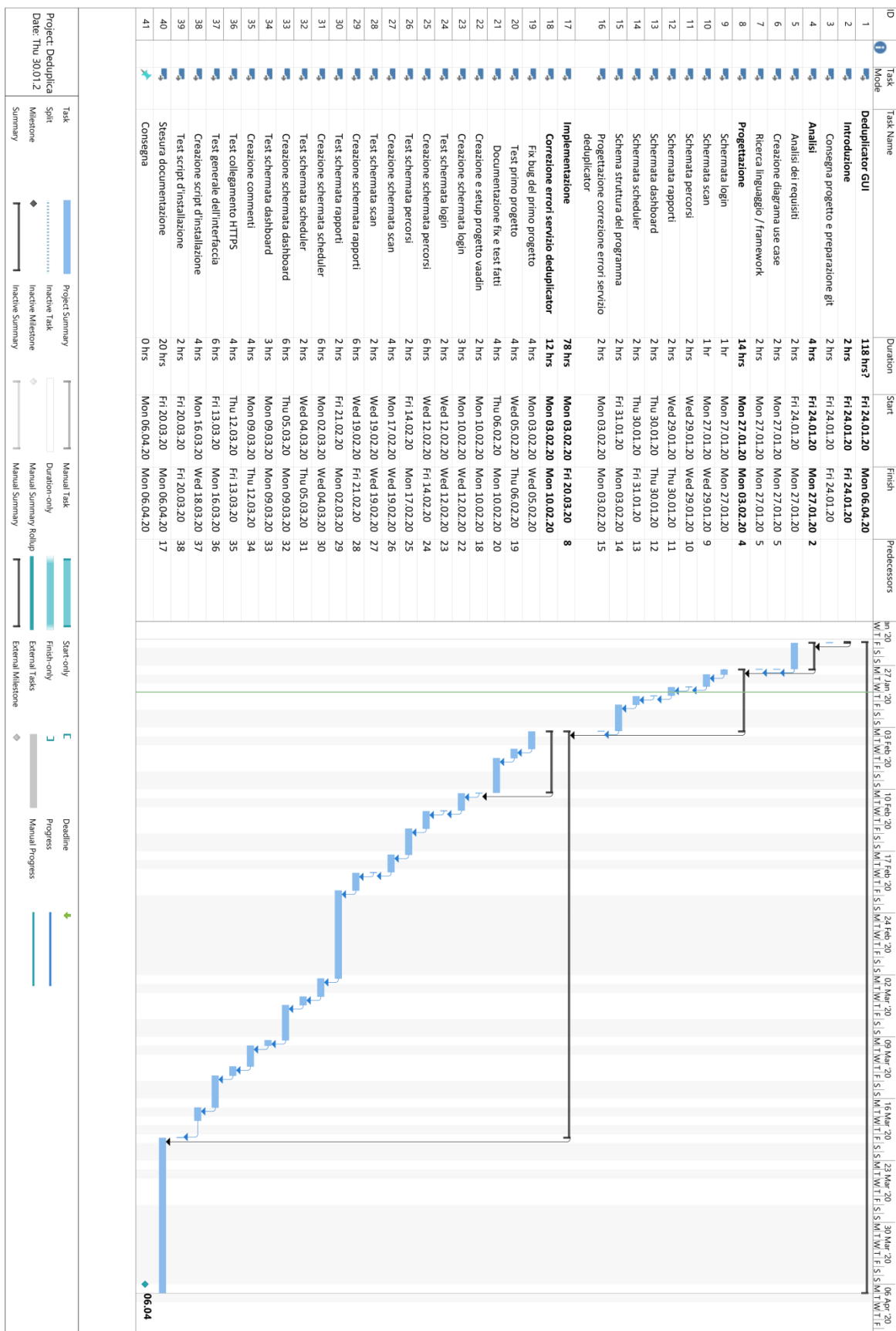


Figura 2 Gantt iniziale

Nella figura 3 si può vedere le macro-sezioni Introduzione e Analisi dove ho pianificato di spendere 6 ore in totale durante i primi giorni del progetto. Nella fase di analisi ho analizzato i requisiti che si trovano nel QdC e gli ho inseriti nella documentazione nella sezione 1.4.2 **Error! Reference source not found..**







2		Introduzione	2 hrs	Fri 24.01.20	Fri 24.01.20	
3		Consegna progetto e preparazione git	2 hrs	Fri 24.01.20	Fri 24.01.20	
4		Analisi	4 hrs	Fri 24.01.20	Mon 27.01.20	2
5		Analisi dei requisiti	2 hrs	Fri 24.01.20	Mon 27.01.20	
6		Creazione diagrama use case	2 hrs	Mon 27.01.20	Mon 27.01.20	5
7		Ricerca linguaggio / framework	2 hrs	Mon 27.01.20	Mon 27.01.20	5

Figura 3 Sezione analisi del Gantt

Nella figura 4 si può vedere in che ordine sarà la durata della creazione dei mockup delle schermate che verranno implementate, in totale ho stimato che mi serviranno 12 ore per la creazione dei mockup delle interfacce e della struttura del programma e altre 2 per progettare le modifiche da fare sul progetto del primo progetto.










8		Progettazione	14 hrs	Mon 27.01.20	Mon 03.02.20	4
9		Schermata login	1 hr	Mon 27.01.20	Mon 27.01.20	
10		Schermata scan	1 hr	Mon 27.01.20	Wed 29.01.20	9
11		Schemata percorsi	2 hrs	Wed 29.01.20	Wed 29.01.20	10
12		Schermata rapporti	2 hrs	Wed 29.01.20	Thu 30.01.20	11
13		Schermata dashboard	2 hrs	Thu 30.01.20	Thu 30.01.20	12
14		Schermata scheduler	2 hrs	Thu 30.01.20	Fri 31.01.20	13
15		Schema struttura del programma	2 hrs	Fri 31.01.20	Mon 03.02.20	14
16		Progettazione correzione errori servizio deduplicator	2 hrs	Mon 03.02.20	Mon 03.02.20	15

Figura 4 Sezione Progettazione del Gantt

Nella figura 5 si può vedere la macro sezione dell'implementazione che è stimata di durare 78 ore, quindi la maggior parte del progetto. All'interno di essa le prime 12 ore ho pianificato che servono a effettuare delle modifiche sul progetto del primo semestre. Queste modifiche riguardano la correzione del orario dello scheduler e il modo in cui vengono fatte partire le thread per la scansione dei percorsi, inoltre le modifiche che andrò a fare per migliorare la scansione mi permetteranno anche di poter ricavare lo stato e il progresso della scansione.

L'implementazione vera e propria inizierà solo dopo che le modifiche del progetto del primo semestre saranno finite.

Visto che il progetto dovrà essere modulare anche i test verranno eseguiti man mano che vengono implementate le diverse parti del progetto, per questo motivo non c'è nessuna macro sezione di test alla fine della progettazione.

Le ultime due attività descrivono il tempo necessario che penso di impegnare per creare uno script di installazione e una guida, che è di 6 ore in totale.
























17		Implementazione	78 hrs	Mon 03.02.20	Fri 20.03.20	8
18		Correzione errori servizio deduplicator	12 hrs	Mon 03.02.20	Mon 10.02.20	
19		Fix bug del primo progetto	4 hrs	Mon 03.02.20	Wed 05.02.20	
20		Test primo progetto	4 hrs	Wed 05.02.20	Thu 06.02.20	19
21		Documentazione fix e test fatti	4 hrs	Thu 06.02.20	Mon 10.02.20	20
22		Creazione e setup progetto vaadin	2 hrs	Mon 10.02.20	Mon 10.02.20	18
23		Creazione schermata login	3 hrs	Mon 10.02.20	Wed 12.02.20	22
24		Test schermata login	2 hrs	Wed 12.02.20	Wed 12.02.20	23
25		Creazione schermata percorsi	6 hrs	Wed 12.02.20	Fri 14.02.20	24
26		Test schermata percorsi	2 hrs	Fri 14.02.20	Mon 17.02.20	25
27		Creazione schermata scan	4 hrs	Mon 17.02.20	Wed 19.02.20	26
28		Test schermata scan	2 hrs	Wed 19.02.20	Wed 19.02.20	27
29		Creazione schermata rapporti	6 hrs	Wed 19.02.20	Fri 21.02.20	28
30		Test schermata rapporti	2 hrs	Fri 21.02.20	Mon 02.03.20	29
31		Creazione schermata scheduler	6 hrs	Mon 02.03.20	Wed 04.03.20	30
32		Test schermata scheduler	2 hrs	Wed 04.03.20	Thu 05.03.20	31
33		Creazione schermata dashboard	6 hrs	Thu 05.03.20	Mon 09.03.20	32
34		Test schermata dashboard	3 hrs	Mon 09.03.20	Mon 09.03.20	33
35		Creazione commenti	4 hrs	Mon 09.03.20	Thu 12.03.20	34
36		Test collegamento HTTPS	4 hrs	Thu 12.03.20	Fri 13.03.20	35
37		Test generale dell'interfaccia	6 hrs	Fri 13.03.20	Mon 16.03.20	36
38		Creazione script d'installazione	4 hrs	Mon 16.03.20	Wed 18.03.20	37
39		Test script d'installazione	2 hrs	Fri 20.03.20	Fri 20.03.20	38

Figura 5 Sezione implementazione Gantt

Nella figura 6 si può vedere l'ultima parte del progetto dove dedico le ultime 20 ore dei progetti alla stesura della documentazione.

40		Stesura documentazione	20 hrs	Fri 20.03.20	Mon 06.04.20	17
41		Consegna	0 hrs	Mon 06.04.20	Mon 06.04.20	

Figura 6 Sezione finale Gantt

3.1 Analisi dei mezzi

Per la creazione di questo progetto ho accesso a tutti i programmi che sono messi a disposizione dalla scuola e 1 accesso presso l'hosting interno nel caso che sia necessario caricare il progetto sull'FTP della scuola.

3.1.1 Software

- Vaadin 14
- VSCode 1.41.1
- MySQL 8.0
- Java 11
- Apache Maven 3.6.1
- Postman 7.17.0
- Node 10.19
- Npm 6.4.1
- Spring boot 2.2.2.RELEASE

3.1.2 Hardware

Per lo sviluppo verrà utilizzato il mio portatile personale che ha le seguenti specifiche:

HP Pavilion 15-0800nz

CPU: i7-8550U

RAM: 16 GB DDR4

OS: Pop!_OS 19.10 / Kernel: 5.3.0-7625-generic

4 Progettazione

Per lo sviluppo della GUI utilizzerò il framework Vaadin che permette di creare una progressive webapp con una interfaccia moderna utilizzando codice java, le interfacce create sono responsive e funzionano completamente su la maggior parte dei browser

4.1 Design dell'architettura del sistema

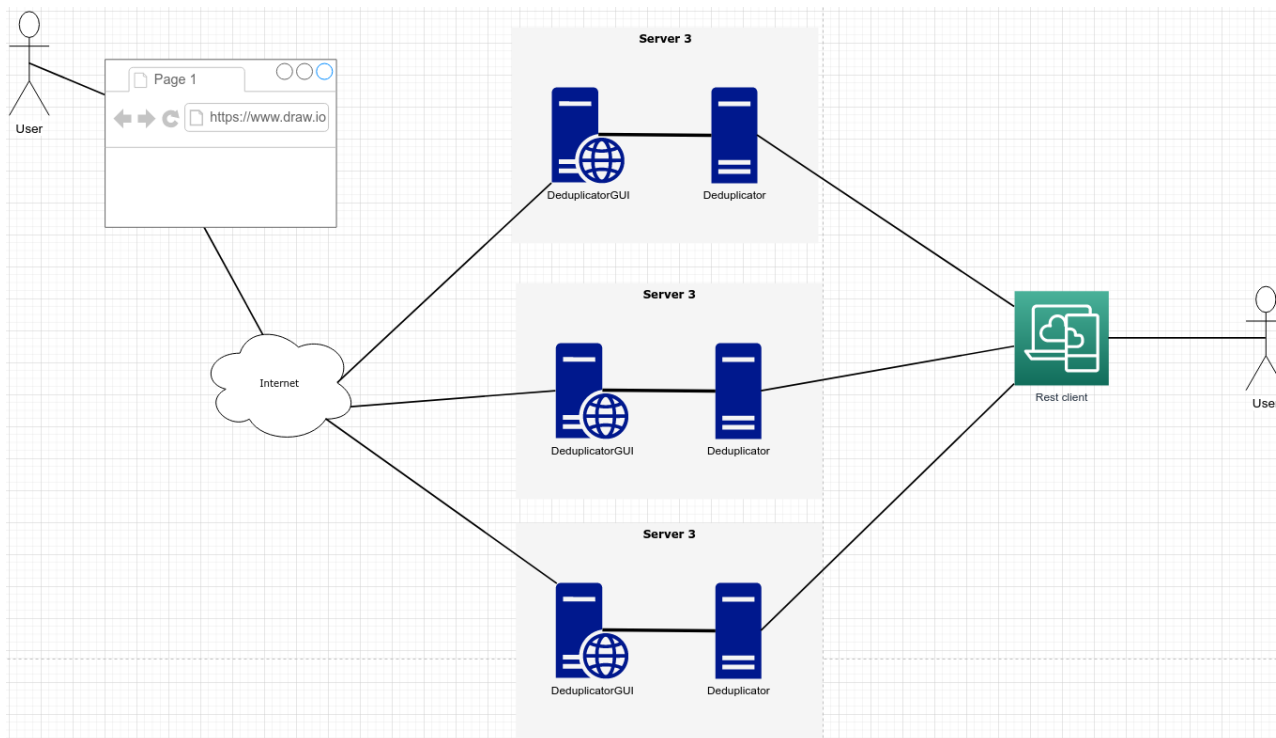


Figura 7 Schema dell'architettura del sistema

•

4.2 Modifiche da apportare al progetto Deduplicator

Ci sono alcune modifiche che sono da eseguire sul vecchio progetto per correggere il funzionamento di esso. Queste modifiche sul progetto del primo semestre includono:

- La correzione del modo in cui viene fatta la scansione
- La possibilità di ricavare lo stato della scansione
- La correzione del tempo nello scheduler poiché adesso l'orario e la data d'esecuzione delle scansioni pianificate è sfasato.

4.2.1 Modifiche thread di scansione

Attualmente il servizio deduplicator esegue la scansione di file nei percorsi impostati grazie a 10 thread di scansione. Queste thread dovrebbero eseguire la scansione dei percorsi in parallelo una con l'altra, ma questo non è il caso. Le thread aspettano che la thread che è stata eseguita prima di essa sia finita, rendendo la scansione sequenziale e non parallela.

Bisogna fare in modo che ci sia una thread che cerca i file nei percorsi specificati dall'utente e gli aggiunge ad una coda e poi ci saranno 10 thread che lavoreranno a svuotare quella coda prendendo quelli file e ricavano tutte le informazioni necessari (hash, data di modifica, grandezza...), inoltre le 10 thread salveranno i file e le loro informazioni nel database.

4.2.2 Ricavare lo stato della scansione

Questa modifica è legata a quella menzionata sopra. Si tratta di ricavare lo stato della scansione grazie alla thread di scansione che verrà implementata. La thread di scansione mi darà l'informazione su quanti file ci sono e grazie al numero di file che ci sono nel database si può ricavare quanti file sono stati scansionati e quanti ne rimangono ancora da scansionare.

4.2.3 Correzione del tempo di esecuzione dello scheduler

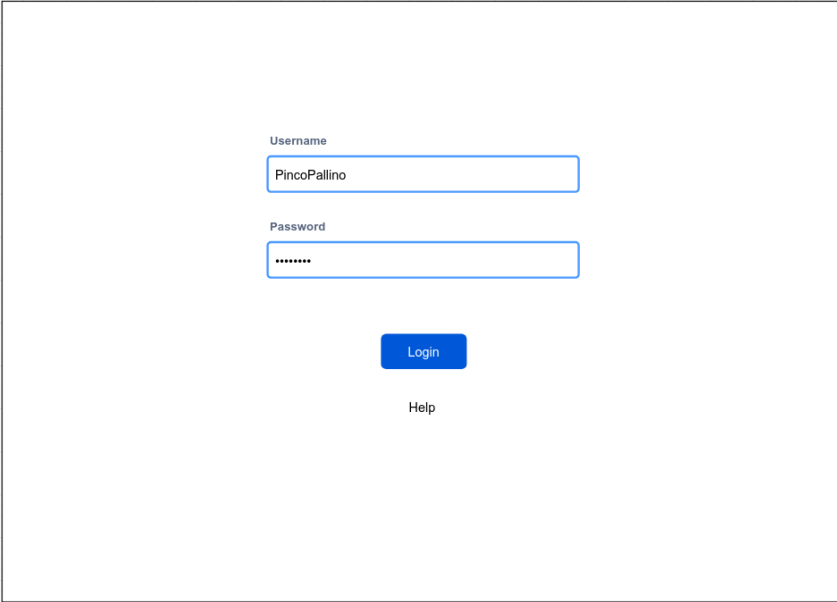
Al momento nel file **ScheduleChecker.java** sulle righe 79 e 80 ci sono due istruzioni che modificano il tempo di esecuzione de impostare alla thread di scansione per le scansioni pianificate:

```
79          startCalendar.add(Calendar.MONTH, 1); // Correzione del Calendario
80          startCalendar.add(Calendar.HOUR_OF_DAY, 2); // Correzione dell'ora
```

Durante l'implementazione del vecchio programma ho aggiunto queste due righe per correggere il mese di un mese in avanti l'ora di due ore in avanti, visto che l'implementazione di questa parte del programma è avvenuta in dicembre dell'anno scorso la correzione del mese era inserita per correggere il mese dal valore di 11 a 12, dopo ho scoperto che il range del mese nel tipo *Calendar* di java va da 0 a 11. Il motivo dietro la correzione dell'orario è una differenza nel fuso orario e il cambiamento da ora legale a ora solare che non ho previsto.

4.3 Design delle interfacce

Nella figura 8 si può vedere come verrà strutturata la schermata di login, per accedere all'interfaccia utente.



The login screen design is centered and contains the following elements:

- A label "Username" above a text input field containing the text "PincoPallino".
- A label "Password" above a password input field containing eight asterisks "*****".
- A blue button labeled "Login" positioned below the password field.
- A text link labeled "Help" positioned below the "Login" button.

Figura 8 Schermata login

Nella figura 9 si può vedere la schermata dell'inserimento dei percorsi, che diventa accessibile insieme a tutte le altre schermate dopo che l'utente effettua il login. L'utente ha la possibilità di inserire un percorso che può essere impostato da scansionare o da ignorare durante una scansione. Cliccando l'icona della cartella si aprirà una visuale modale, che conterrà un file browser del filesystem del server, rappresentata nella Figura 10. Nella metà inferiore della finestra ci sarà una lista di percorsi già salvati sul server, essi si possono eliminare o modificare.

Path	Type	Manage
/home/john/Documents/Backup/filename.pdf	Ignore	DELETE MODIFY
/home/john/Documents/Backup/otherfile.pdf	Ignore	DELETE MODIFY
/home/john/Documents/Backup/bigFile.pdf	Ignore	DELETE MODIFY
/home/john/Documents/full_Folder	Scan	DELETE MODIFY
/home/john/Documents/folder_A/	Ignore	DELETE MODIFY
/home/john/Desktop/	Scan	DELETE MODIFY

Figura 9 Schermata dell'inserimento dei percorsi

```

Select path or file
> settings
  > templates
  > tests
    package.json
    file.txt
    debug.py
> templates
> tests
  package.json
  file.txt
  debug.py
  
```

Figura 10 Schermata del file browser per la selezione del percorso

Nella figura 11 è rappresentata la schermata della gestione delle scansioni. L'utente può avviare, fermare o mettere in pausa una scansione.

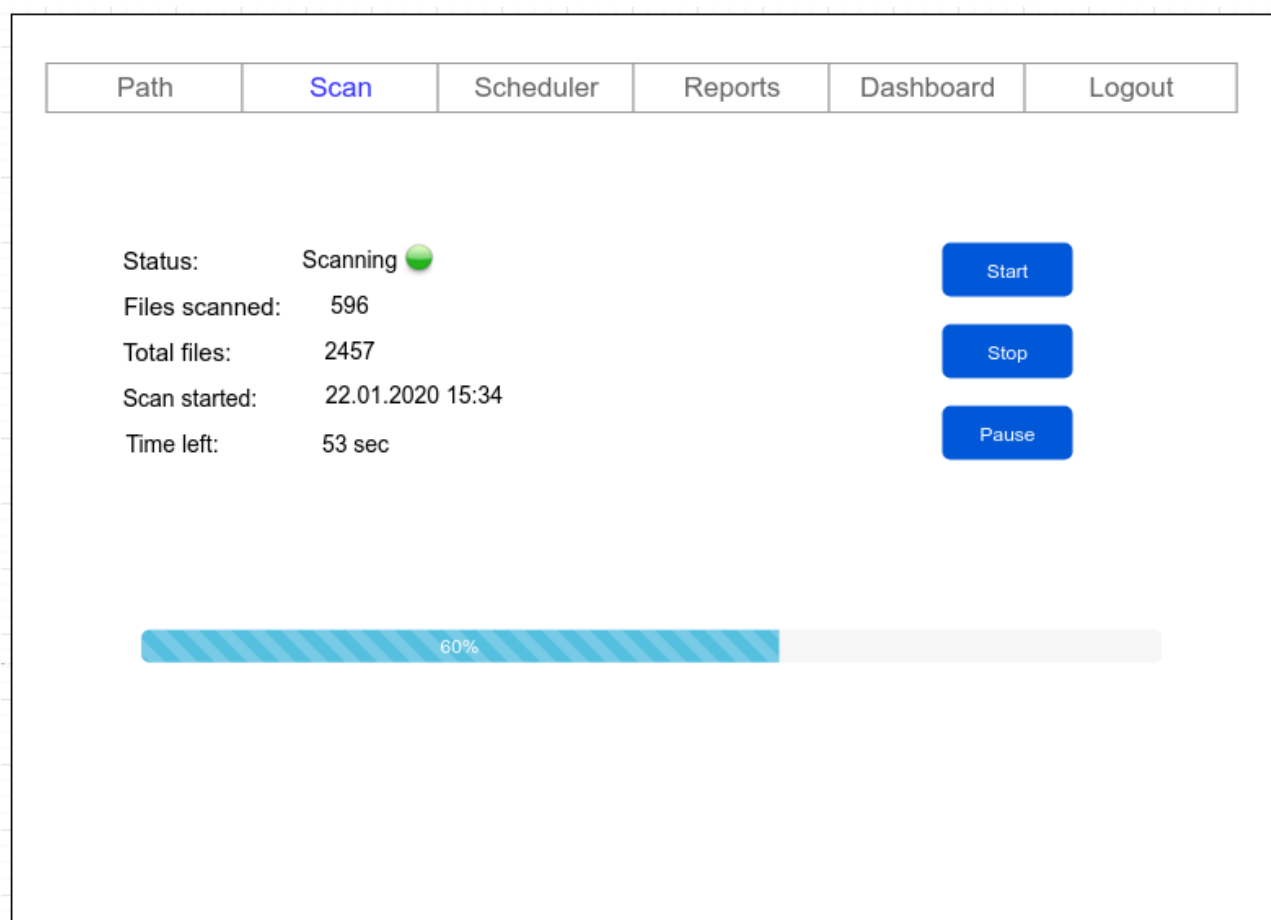



Figura 11 Schermata della gestione delle scansioni

Nella figura 12 si può vedere la schermata dello scheduler, qua si può impostare la data di una futura scansione oppure la data sulla quale verrà eseguita settimanalmente o mensilmente una scansione.

Path	Scan	Scheduler	Reports	Dashboard	Logout
------	------	------------------	---------	-----------	--------

Data:

Today


<
Gennaio 2020
>

SUN	MON	TUE	WED	THU	FRI	SAT
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

Permissions

☐ One off
☒ Daily
☐ Weekly
☐ Monthly

Set scheduler

Figura 12 Schermata della gestione dello scheduler

Nella figura 13 si possono vedere i rapporti delle scansioni fatte, cliccando il bottone “i”, blue, accanto al dropdown menu si apre una visuale con le informazioni riguardo la scansione scelta.

Nella tabella sotto il dropdown menu si possono vedere i duplicati trovati in quella tabella. Un duplicato è rappresentato dalla barra grigia con informazioni riguardo a quel duplicato (numero di file, grandezza di ogni file e hash di contenuti dei file), sotto la barra grigia si possono vedere i percorsi dei file duplicati con la data della loro ultima modifica e le azioni che si possono fare su quel file (ignorare, eliminare o muovere). Inoltre c'è la possibilità di ignorare tutti i file di quel duplicato.

Path	Scan	Scheduler	Reports	Dashboard	Logout
------	------	-----------	----------------	-----------	--------

Scan 15.09.2019 11:30

i

Number of files: 4
Size: 78 KB
Hash: 7215ee9c7d9dc229d2921a40e899ec5f
Ignore all duplicates

Path	Last modified	Manage		
/home/john/Documents/Backup/filename.pdf	2019-10-11	IGNORE	DELETE	MOVE
/home/john/Documents/Backup/otherfile.pdf	2016-07-18	IGNORE	DELETE	MOVE
/home/john/Documents/Backup/bigFile.pdf	2020-01-13	IGNORE	DELETE	MOVE
/home/john/Documents/folder_A/bigFileDup.pdf	2019-12-14	IGNORE	DELETE	MOVE

Number of files: 6
Size: 2.3 MB
Hash: 7815696ecbf1c96e6894b779456d330e
Ignore all duplicates

Path	Last modified	Manage		
/home/john/Documents/Backup/filename.pdf	2019-10-11	IGNORE	DELETE	MOVE
/home/john/Documents/Backup/otherfile.pdf	2016-07-18	IGNORE	DELETE	MOVE
/home/john/Documents/Backup/bigFile.pdf	2020-01-05	IGNORE	DELETE	MOVE
/home/john/Documents/folder_A/bigFileDup.pdf	2019-12-14	IGNORE	DELETE	MOVE

Show 15 operations to be executed

Apply

Figura 13 Schermata della gestione dei rapporti e duplicati

Nella figura 14 si può vedere la pagina di gestione del servizio e della GUI. In questa schermata l'utente ha la possibilità di cambiare la password, nome utente, le credenziali per il server MySQL del servizio e la posizione del file di log. Inoltre, si può impostare l'intervallo di tempo nel quale verrà aggiornato lo stato della scansione rappresentato tramite la barra e i dati sulla parte sinistra della schermata nella figura 11.

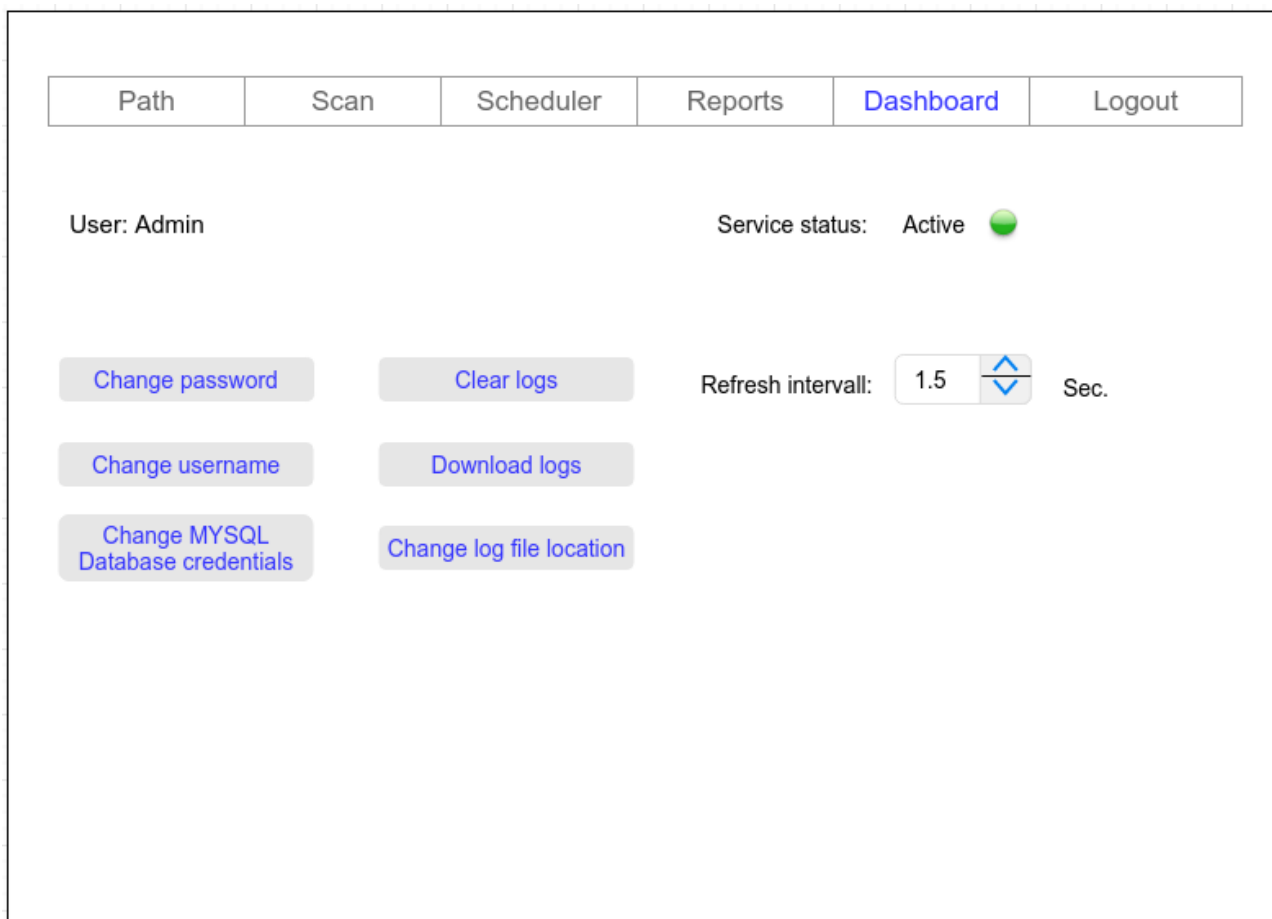


Figura 14 Schermata della gestione del servizio e della GUI

5 Implementazione

Per il versioning del progetto è stato usato GitHub Flow che consiste nel creare una branch di sviluppo (develop) e una principale (master). Usando questo metodo gli aggiornamenti, risoluzione dei problemi e aggiunte di funzionalità vengono caricate sulla branch di sviluppo mentre sulla branch principale vengono caricate solo le nuove funzionalità delle quali è stato verificato il funzionamento.

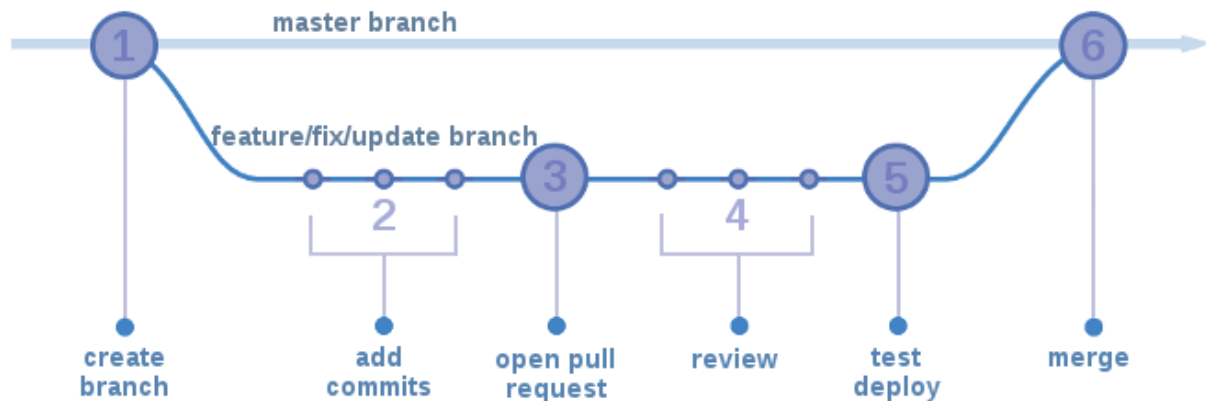


Figura 15 Diagramma Github Flow

5.1 Framework Vaadin

Il framework utilizzato per lo sviluppo di questo progetto si chiama Vaadin.

Vaadin è un framework open source fatto per gli sviluppatori di java per facilitare la creazione di UI o applicazioni web progressive moderne, scrivendo tutto con codice JAVA senza dover usare i linguaggi standard tipo HTML, CSS o JS.

Le applicazioni Vaadin lavorano su un server e gestiscono tutta la comunicazione in modo automatico e sicuro.

Vaadin mette a disposizione dei componenti già pronti per l'utilizzo, con tante funzionalità che permettono la creazione di potenti interfacce con poco codice, inoltre mette a disposizione un repository di componenti creati dalla comunità grazie al Component Builder che permette la creazione di componenti con funzionalità specifiche.

Per lo sviluppo di questo progetto è stato usato Vaadin 14 che è l'ultima versione LTS.

Il progetto è stato creato grazie al tool disponibile sul seguente sito di Vaadin:

<https://vaadin.com/start/v14>.

Il progetto creato si basa su Spring boot, utilizza fatto con java 11 e Maven come gestore di dipendenze.

5.1.1 Maven

Maven è uno strumento di automazione della compilazione utilizzato principalmente per progetti Java.

L'automazione della compilazione è il processo di automazione della creazione di un eseguibile del software e dei processi associati, tra cui: compilazione del codice sorgente del computer in codice binario, impacchettamento del codice binario ed esecuzione di test automatizzati.

Maven scarica dinamicamente le librerie Java e i plug-in Maven da uno o più repository come il Repository centrale Maven 2 e li archivia in una cache locale. Questa cache locale di artefatti scaricati può anche essere aggiornata con artefatti creati da progetti locali.

5.2 Correzione errori primo progetto

5.2.1 Nuovo metodo di scansione

Nel vecchio modo in cui veniva fatta la scansione il servizio faceva partire la classe ScanManager che per ogni percorso inserito dall'utente faceva partire un pool di 10 Thread (ScannerThread) che scansionava tutti i figli del percorso impostato. A questo punto ogni Thread aveva una lista dei figli del suo percorso impostato. Tutti i figli che sono file venivano salvati in un'altra lista, mentre per tutti i figli che sono una cartella veniva subito fatto partire uno ScannerThread per scansionare così in modo ricorsivo tutti i file dal percorso iniziale in giù. Alla fine della scansione veniva fatto partire un pool di 10 Thread (Hasher) che ricavano tutti i dati (hash, data dell'ultima modifica e grandezza) sui file trovati.

L'esecuzione dei pool era fatta in modo sbagliato perché non veniva impostata una thread che lavora su un percorso solo da permettere alla pool di gestire l'esecuzione, invece venivano fatte partire le thread che iteravano su una lista di percorsi rendendo le pool inutili e la scansione quasi in modo seriale.

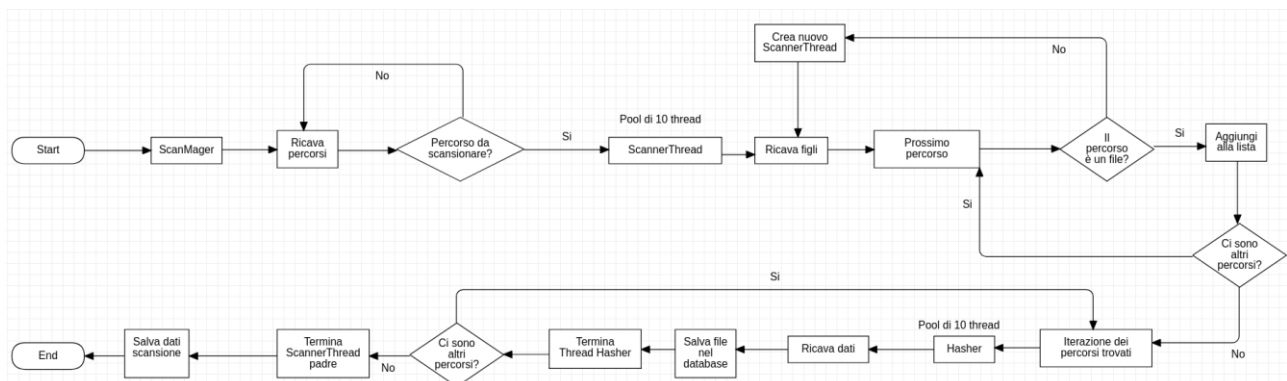


Figura 16 Diagramma di flusso rappresentante il vecchio modo in cui veniva fatta la scansione

Nel nuovo metodo la scansione parte con la creazione di una thread (FilesScanner) che trova tutti i file che ci sono nei percorsi specificati dall'utente dopodiché viene creato il pool di esecuzione delle thread (ScannerWorker) che ricavano i dati dai percorsi trovati. Nel pool viene impostato il numero massimo di thread che possono lavorare contemporaneamente, e poi vengono subito fatte partire con l'aggiunta del primo percorso da scansionare. Visto che i percorsi sono di più rispetto alle thread, queste operazioni vengono messe in coda e gestite automaticamente dalla pool.

Ogni ScannerWorker sé può salva il proprio file nel database.

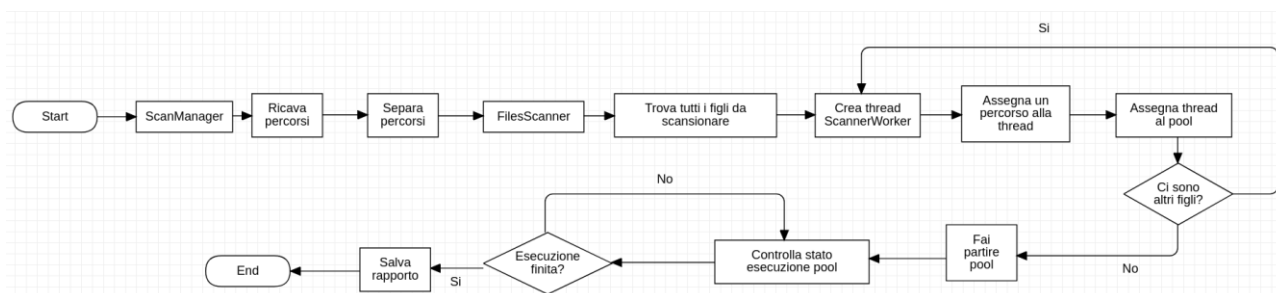


Figura 17 Diagramma di flusso rappresentante il nuovo modo in cui viene fatta la scansione

5.2.2 Ottenimento stato della scansione

Per l'ottenimento dello stato della scansione viene utilizzata una thread che ogni 200 ms. richiama il metodo *calcualeProgress()* che calcola quanti file sono stati trovati dal *FilesScanner*, quanti file sono nel sono nel database, quanti non sono stati salvati (per via di errore o altro) e fa il rapporto tra quest'ultimi, inoltre stampa su terminale lo stato della scansione.

```

statusThread = new Thread() {
    @Override
    public void run() {
        try {
            while (!isInterrupted() && scanProgress < 1f) {
                synchronized (statusMonitor) {
                    if (paused) {
                        statusMonitor.wait();
                    }
                }
                System.out.print("\rProgress: " + calcuateProgress() + "%");
                synchronized (this) {
                    this.wait(POLLING_DELAY);
                }
            }
        } catch (InterruptedException ie) {
            System.out.print("\rProgress: " + calcuateProgress() + "%\n");
        }
    }
};
  
```

Il calcolo effettuato è il seguente: $1 - (\text{file trovati} - \text{numero di file nel database} - \text{salvataggi che non sono andati a buon fine}) / \text{file trovati}$

Lo stato della scansione viene salvato nella variabile globale *scanProgress* e il metodo ritorna una stringa formattata come percentuale con 2 cifre dopo la virgola.

```
private String calculateProgress() {
    if (totalFiles != 0) {
        scanProgress = (1f
            - (((float) totalFiles - (float) fileRepository.findByReport(report) -
                getUnsuccessfulSaves())
                / (float) totalFiles));
    } else {
        scanProgress = -1;
    }
    return String.format(java.util.Locale.getDefault(), "%.2f", scanProgress * 100f);}
}
```

5.2.3 Correzione tempo scheduler

Per risolvere il problema della data e dell'ora dello scheduler ho semplicemente rimosso le righe 80 e 81 che contenevano il seguente codice:

```
startCalendar.add(Calendar.MONTH, 1); // Correzione del Calendario
startCalendar.add(Calendar.HOUR_OF_DAY, 2); // Correzione dell'ora
```

5.3 Layout generale dell'applicazione

Il layout della GUI si ispira a quello usato nell'applicazione di esempio che si trova sul sito ufficiale del framework Vaadin: <https://vaadin.com/start/1/s/simple-ui>. Nell'applicazione di esempio viene utilizzato l'**AppLayout** che implementa un semplice ma funzionale menu che si trova sul lato sinistro della schermata.

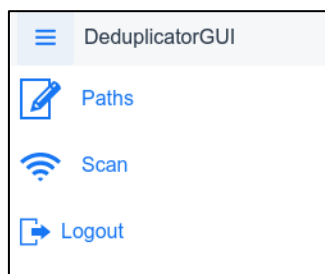


Figura 18 Menu di lato aperto

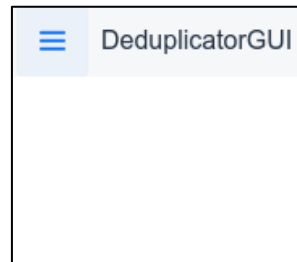


Figura 19 Menu di lato chiuso

5.3.1 MainLayout

Il layout di base della applicazione viene implementato nella classe *MainLayout* che estende la classe *AppLayout*.

Nel costruttore viene creato il tasto dell'apertura e chiusura del menu, la barra orizzontale in alto alla pagina e tutti gli elementi del menu incluso il tasto di Logout. Tutte le view si basano su questo layout e quindi l'applicazione avrà un aspetto comune tra tutte le view.

```
public class MainLayout extends AppLayout{
    public MainLayout() {
        //Tasto apertura/chiusura menu
        final DrawerToggle drawerToggle = new DrawerToggle();
        drawerToggle.addClassName("menu-toggle");
        addToNavbar(drawerToggle);

        //Barra orizzontale in alto con il titolo
        final HorizontalLayout top = new HorizontalLayout();
        top.setDefaultVerticalComponentAlignment(Alignment.CENTER);
        top.setClassName("menu-header");
        final Label title = new Label("DeduplicatorGUI");
        top.add(title);
        addToNavbar(top);

        //Gli oggetti del menu
        addToDrawer(createMenuLink(MainView.class, MainView.VIEW_NAME, VaadinIcon.HOME.create()));
        addToDrawer(createMenuLink(PathView.class, PathView.VIEW_NAME, VaadinIcon.EDIT.create()));
        addToDrawer(createMenuLink(ScanView.class, ScanView.VIEW_NAME, VaadinIcon.BUG.create()));
        addToDrawer(createMenuLink(ReportView.class, ReportView.VIEW_NAME,
VaadinIcon.FILE_TEXT.create()));
        addToDrawer(createMenuLink(SchedulerView.class, SchedulerView.VIEW_NAME,
VaadinIcon.CALENDAR.create()));
        addToDrawer(createMenuLink(DashboardView.class, DashboardView.VIEW_NAME,
VaadinIcon.DASHBOARD.create()));
        logoutButton = createMenuButton("Logout", VaadinIcon.SIGN_OUT.create());
        logoutButton.addClickListener(e -> logout());
        logoutButton.getElement().setAttribute("title", "Logout (Ctrl+L)");
    }
}
```

5.3.2 Client

Il client usato per la comunicazione tra il client web e le REST API è stato implementato usando il *RestTemplate* di Spring. Questa classe è già stata sviluppata nel progetto scorso (deduplicator), ma c'erano parti mancanti o che erano testate, non avendo finito il progetto scorso.

Si seguito verranno descritte le parti più importanti di quella classe.

Il metodo *init* viene chiamato per inizializzare l'oggetto client, esso accetta come parametri il nome utente e la password dell'utente i quali vengono salvati e utilizzati per fare l'autenticazione per tutte le richieste in seguito nel metodo *createHeaders*. Il metodo carica anche la chiave per l'autenticazione SSL e crea l'oggetto *RestTemplate* che verrà utilizzato per fare tutte le richieste alle API.

Il metodo *init* non viene chiamato dal costruttore per via della creazione automatica della classe da parte del framework spring che richiede un costruttore vuoto.

```
public boolean init(String username, String password) {
    this.username = username;
    this.password = password;
    try {
        FileInputStream in = new FileInputStream(new File("deduplicator.p12"));
        String caPassword = new String(Base64.getDecoder().decode(props.getCAPassword()));
        try {
            keyStore = KeyStore.getInstance("PKCS12");
            keyStore.load(in, caPassword.toCharArray());
        } catch (IOException | KeyStoreException | NoSuchAlgorithmException |
            CertificateException e) {
            Logger.getGlobal().log(Level.SEVERE, "Unable to load SSL key into HTTPS client");
            e.printStackTrace(System.out);
        }
        try {
            TrustStrategy acceptingTrustStrategy = (X509Certificate[] chain,
                String authType) -> true;
            SSLContext sslContext = SSLContextBuilder.create().loadKeyMaterial(keyStore,
                caPassword.toCharArray())
                .loadTrustMaterial(null, acceptingTrustStrategy).build();

            HttpClient httpClient = HttpClients.custom().setSSLContext(sslContext).build();

            HttpComponentsClientHttpRequestFactory requestFactory =
                new HttpComponentsClientHttpRequestFactory();

            requestFactory.setHttpClient(httpClient);
            restTemplate = new RestTemplate(requestFactory);
            return true;
        } catch (UnrecoverableKeyException | NoSuchAlgorithmException | KeyStoreException
            | KeyManagementException e) {
            Logger.getGlobal().log(Level.SEVERE, "Unable to create client: " +
                e.getMessage());
            e.printStackTrace();
            return false;
        }
    } catch (FileNotFoundException fne) {
        Logger.getGlobal().log(Level.SEVERE, "CA certificate not found");
        return false;
    }
}
```

5.3.2.1 Metodo per controllo dell'autenticazione

Il metodo per il controllo dell'autenticazione si chiama *isAuthenticated* esso accetta due parametri di tipo String, che sono l'indirizzo ip e porta sulla quale si tenterà di eseguire fare l'accesso. Le credenziali per l'accesso sono impostate alla creazione dell'oggetto Client nel metodo *init*.

```
public HttpStatus isAuthenticated(String host, int port) throws RestClientException {

    HttpEntity<Map<String, String>> requestEntity = new HttpEntity<>(createHeaders(false));
    ResponseEntity<String> response = null;
    try {
        response = restTemplate.exchange(prefix + host + ":" + port + "/access/login/",
            HttpMethod.GET, requestEntity,
            String.class);
    } catch (RestClientException rce) {
        Logger.getLogger().log(Level.SEVERE, "Rest client exception: " + rce.getMessage());
        if (rce.getMessage().startsWith("I/O error on GET request")) {
            return HttpStatus.EXPECTATION_FAILED;
        }
        if (rce.getMessage().strip().startsWith("401")) {
            return HttpStatus.UNAUTHORIZED;
        }
    }
    if (response != null) {
        if (response.getStatusCode().equals(HttpStatus.OK)) {
            this.host = host;
            setPort(port);
        }
        return response.getStatusCode();
    } else {
        return HttpStatus.SERVICE_UNAVAILABLE;
    }
}
```

5.3.2.2 Metodo che imposta l'header della richiesta

Il metodo *createHeaders* crea gli header da aggiungere alla richiesta. Gli header includono l'autenticazione BASIC. La variabile *hasFormData* passata come parametro indica se gli header descrivono una richiesta che contiene dei dati, nel caso il parametro sia true, viene impostato il parametro *Content-Type* su *multipart/form-data* che indica il contenuto presente nella richiesta fa parte di un form.

```
private HttpHeaders createHeaders(boolean hasFormData) {
    HttpHeaders header = new HttpHeaders();
    String auth = username + ":" + password;
    byte[] encodedAuth = Base64.getEncoder().encode(
        auth.getBytes(StandardCharsets.US_ASCII));
    String authHeader = "Basic " + new String(encodedAuth);
    header.add("Authorization", authHeader);
    if (hasFormData) {
        header.setContentType(MediaType.MULTIPART_FORM_DATA);
    }
    return header;
}
```

5.3.2.3 Richieste GET

Per effettuare tutte le richieste di tipo GET viene utilizzato il metodo *get*.

Accetta come parametro il percorso sul quale verrà effettuata la richiesta, per effettuare la richiesta viene utilizzato il metodo *getForEntity* della classe *RestTemplate* che restituisce un oggetto di tipo *ResponseEntity* nel caso che la richiesta va a buon fine, altrimenti viene tirata una eccezione, la risposta può essere utilizzata per ricavare lo stato e il body della risposta.

Nel caso di un errore viene ritornato il valore *null*.

```
public ResponseEntity<String> get(String path) {
    HttpEntity<Map<String, String>> requestEntity = new HttpEntity<>(createHeaders(false));
    try {
        ResponseEntity<String> response = restTemplate.getForEntity(prefix + host + ":" +
port + "/" + path, String.class, requestEntity);
        if (response.getStatusCode().equals(HttpStatus.OK)) {
            return response;
        } else {
            Logger.getLogger().log(Level.SEVERE, "Response status code is not OK");
            return null;
        }
    } catch (RestClientException rce) {
        Logger.getLogger().log(Level.SEVERE, "Rest client exception: " + rce);
    }
    return null;
}
```

5.3.2.4 Richieste POST

Per effettuare tutte le richieste di tipo POST viene utilizzato il metodo **post**.

Accetta come parametro il percorso sul quale verrà effettuata la richiesta e i valori che verranno mandati insieme alla richiesta, per effettuare la richiesta viene utilizzato il metodo *exchange* della classe *RestTemplate* che restituisce un oggetto di tipo *ResponseEntity* nel caso che la richiesta va a buon fine, altrimenti viene tirata una eccezione, la risposta può essere utilizzata per ricavare lo stato e il body della risposta.

I valori passati come parametro sono di tipo *MultiValueMap* che salva i dati in un formato di tipo chiave – valore.

Per riuscire a mandare i dati passati come parametro insieme alla richiesta, viene usato il metodo *createHeaders(true)* alla creazione dell'oggetto *requestEntity*.

Nel caso di un errore viene ritornato il valore *null*.

Il metodo post viene utilizzato dalla classe *ScanView* per avviare, fermare, mettere in pausa e proseguire una scansione; viene utilizzato anche dalla classe *AccessControl* per effettuare il logout quando viene premuto il tasto *logout* nel menu di lato oppure dopo aver cambiato la password o username.

```
public ResponseEntity<String> post(String path, MultiValueMap<String, Object> values) {
    values = values == null ? new LinkedMultiValueMap<>() : values;
    HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(values,
                                                                              createHeaders(true));

    ResponseEntity<String> response = null;
    try {
        response = restTemplate.exchange(prefix + host + ":" + port + "/" + path,
                                         HttpMethod.POST, requestEntity, String.class);
    } catch (RestClientException rce) {
        Logger.getGlobal().log(Level.SEVERE, "Rest Client Exception: " + rce.getMessage());
        rce.printStackTrace(System.out);
    }
    return response;
}
```

5.3.2.5 Richieste PUT

Per effettuare tutte le richieste di tipo PUT viene utilizzato il metodo *put*.

Il metodo *put* è simile al metodo *post* accetta e usa i parametri in egual modo come il metodo **post**.

Questo metodo a differenza di quello *post* fa un ulteriore controllo sulla risposta ricevuta, più precisamente se il body della risposta è costruito in formato JSON esso viene trasformato in un *JSONObject* tramite quale viene verificata la presenza del campo *message* che sta ad indicare che la risposta ritornata è un messaggio d'errore e quindi qualcosa è andato storto.

Nel caso di un errore viene ritornato il valore *null*.

Questo metodo viene utilizzato da tutti i metodi che inseriscono dei dati nel database delle API, questi sono: *insertSchedule*, *updatePassword*, *updateUsername*, *insertAction* e *addUser*.

```
public ResponseEntity<String> put(String path, MultiValueMap<String, Object> values) {
    values = values == null ? new LinkedMultiValueMap<>() : values;
    HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(values,
                                                                              createHeaders(true));

    ResponseEntity<String> response = null;
    try {
        response = restTemplate.exchange(prefix + host + ":" + port + "/" + path,
                                         HttpMethod.PUT, requestEntity, String.class);
    } catch (RestClientException rce) {
        try {
            JSONObject resp = (JSONObject) parser.parse(rce.getMessage());
            if (resp.get("message") != null) {
                Logger.getGlobal().log(Level.SEVERE, "Rest client exception with error: " +
                resp.get("message"));
            } else {
                Logger.getGlobal().log(Level.SEVERE, "Rest client exception: unable to pars
                e exception message");
            }
        } catch (ParseException pe) {
            Logger.getGlobal().log(Level.SEVERE, "Rest client exception: unable to parse ex
            ception message");
        }
        Logger.getGlobal().log(Level.SEVERE, "Rest client exception: general error");
    }
    return response;
}
```

5.3.2.6 Richieste DELETE

Le richieste di tipo DELETE vengono eseguite dal metodo *delete*.

Esso come il metodo *put* è simile al metodo *post* nella gestione dei parametri passati.

Viene utilizzato soltanto dal metodo *deletePath* della stessa classe che serve ad eliminare i percorsi dalla *PathView*.

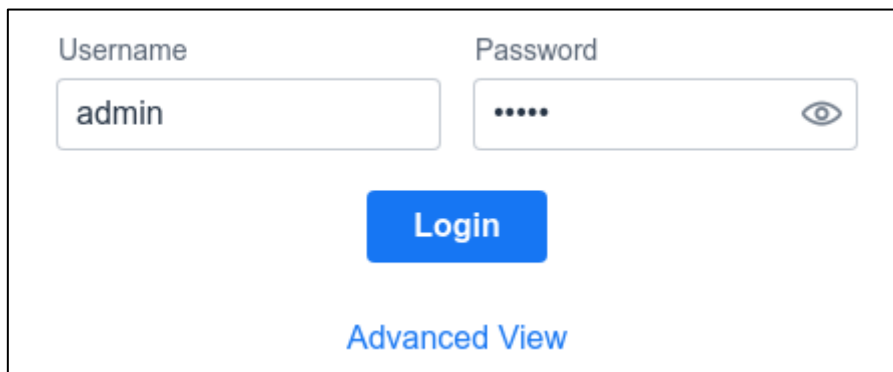
Questo metodo ritorna la risposta ricevuta o nel caso essa sia *null* ritorna un nuovo *ResponseEntity* con lo stato impostato sul codice 400 indicando che la richiesta è malformata.

```
public ResponseEntity<String> delete(String path, MultiValueMap<String, Object> values) {
    HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(values,
                                                                              createHeaders(true));
    ResponseEntity<String> response = null;
    try {
        response = restTemplate.exchange(prefix + host + ":" + port + "/path/",
                                       HttpMethod.DELETE, requestEntity, String.class);
    } catch (RestClientException rce) {
        Logger.getGlobal().log(Level.SEVERE, "Rest Client Exception: " + rce.getMessage());
    }
    return Objects.requireNonNullElseGet(response, () ->
                                         new ResponseEntity<String>(HttpStatus.BAD_REQUEST));
}
```

5.3.3 LoginView

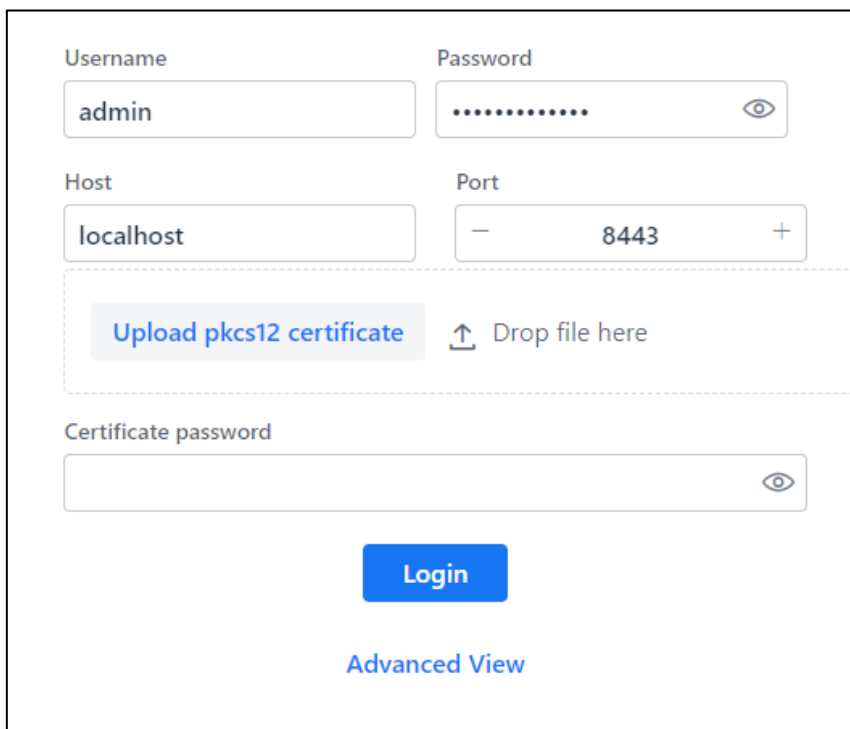
La LoginView viene utilizzata per ricevere le credenziali dall'utente che verranno utilizzate per collegarsi e autenticarsi alle REST API.

Essa è composta da un campo del nome utente e uno della password per permettere le chiamate alle funzioni REST del servizio deduplicator.



The screenshot shows a basic login interface. It has two input fields at the top: 'Username' with the value 'admin' and 'Password' with masked characters '.....'. To the right of the password field is an eye icon for toggling visibility. Below these fields is a blue 'Login' button. At the bottom, there is a blue link labeled 'Advanced View'.

Figura 20 La schermata di login in visuale di base



The screenshot shows an advanced login interface. It includes the 'Username' and 'Password' fields from the basic view. Below these, there are 'Host' and 'Port' fields. The 'Host' field contains 'localhost'. The 'Port' field is a numeric input with a range from '-' to '8443' and a '+' button. Below the host and port fields is a dashed box containing an 'Upload pkcs12 certificate' button and a 'Drop file here' instruction with an upload icon. Below this dashed box is a 'Certificate password' field with an eye icon. At the bottom, there is a blue 'Login' button and a blue link labeled 'Advanced View'.

Figura 21 La schermata di login in visuale avanzata

Tramite la visuale avanzata si possono scoprire due bottoni che permettono all'utente di modificare l'indirizzo e la porta dove verrà effettuato il tentativo di login.

L'indirizzo del host impostato di default è localhost e la porta è 8443.

La comunicazione tra questo e il progetto deduplicator viene fatta tramite l'interfaccia di loopback visto che il servizio e la GUI lavorano sulla stessa macchina, questo impedisce che la comunicazione tra quest'ultimi venga intercettata dall'esterno.

L'utente viene reindirizzato alla LoginView, da qualsiasi altra, se non è stato effettuato il login nella sessione attuale.

La classe implementa le seguenti tre annotazioni:

- Route – Indica a Vaadin che questa classe è raggiungibile tramite l'URL con il valore indicato. Per esempio: *info-4.local.samt/login*
- PageTitle- Imposta il titolo della pagina visibile nel tab del browser.
- CssImport – il percorso del file css del quale verrà caricato lo stile aggiuntivo della pagina.

```
@Route(value = "login")
@PageTitle(value = "Deduplicator - Login")
@CssImport(value = "../styles/login-form-style.css")
public class LoginView extends VerticalLayout {
```

5.3.3.1 Metodo tryLogin

Il metodo tryLogin tenta di fare un login con le credenziali inserite nei campi *username* e *password* al server inserito nel campo *host* e *port*. Dopo aver fatto tutti i controlli sui dati viene creato un nuovo oggetto *client* con il seguente codice: `(Client) context.getBean("connectionClient");`

Dopodiché viene invocato il metodo init per inizializzare l'oggetto *client*.

L'autenticazione viene tentata invocando il metodo isAuthenticated() della classe client, il quale ritorna lo stato della risposta dello server che viene analizzata e nel caso di errore viene mostrata una notifica con il messaggio d'errore appropriato.

```
if (!user.isBlank()) {
    if (Validator.isValidIP(host) || host.equals("localhost")) {
        if (port > 0 && port <= 65535) {
            if (!certificatePassword.getValue().isBlank())
                settings.setCAPassword(Base64.getEncoder().encodeToString(certificatePassword
                    .getValue().getBytes()));
            client = (Client) context.getBean("connectionClient");
            try {
                if (client.init(user, pass)) {
                    HttpStatus resp = client.isAuthenticated(host, port);
                    switch (resp) {
                        case OK:
                            Logger.getGlobal().log(Level.INFO, "User signed in successfully");
                            accessControl.signIn(user, client);
                            UI.getCurrent().navigate("");
                            break;
                        case UNAUTHORIZED:
                            Notification
                                .show("Invalid credentials", settings.getNotificationLength(),
                                    Notification.Position.TOP_END)
                                .addThemeVariants(NotificationVariant.LUMO_ERROR);
                            Logger.getGlobal().log(Level.WARNING, "Invalid credentials");
                            break;
                    }
                }
            } catch (Exception e) {
                // ...
            }
        }
    }
}
```

```

        case SERVICE_UNAVAILABLE:
            Notification
                .show("Server not reachable", settings.getNotificationLength(),
                    Notification.Position.TOP_END)
                .addThemeVariants(NotificationVariant.LUMO_ERROR);
            Logger.getGlobal().log(Level.SEVERE, "Server not reachable");
            break;
        case EXPECTATION_FAILED:
            Notification
                .show("Host not registered as an alias in the certificate, try uploadin
g a new certificate",
                    settings.getNotificationLength(), Notification.Position.TOP_END)
                .addThemeVariants(NotificationVariant.LUMO_ERROR);
            Logger.getGlobal().log(Level.SEVERE, "Host not registered as an alias in th
e certificate");
            break;
        default:
            Notification
                .show("Unknown error occurred", settings.getNotificationLength(),
                    Notification.Position.TOP_END)
                .addThemeVariants(NotificationVariant.LUMO_ERROR);
            Logger.getGlobal().log(Level.SEVERE, "Unknown error occurred");
            break;
    }

```

5.3.4 PathView

La PathView si occupa della gestione dei percorsi del servizio Deduplicator.

Tramite la PathView si possono aggiungere, eliminare e modificare i percorsi presenti sul server. I percorsi possono essere impostati da scansionare o da ignorare.

Questa view ha le stesse annotazioni come la LoginView con valori modificati facendo in modo che raggiungibile tramite l'URL con il valore indicato. Per esempio: *info-4.local.samt/path*

```
@Route(value = "path", layout = MainLayout.class)
@PageTitle(value = "Deduplicator - Path")
@CssImport(value = "../styles/report-view.css")
public class PathView extends VerticalLayout {
```

I metodi principali sono descritti nei sottocapitoli in seguito.

5.3.4.1 Metodo openRootSelect

Il metodo openRootSelect viene chiamato quando l'utente clicca sul input per inserire un percorso da aggiungere. Il metodo controlla se esistono più percorsi di radice del filesystem. Questo metodo esiste per poter selezionare i percorsi di radice sui sistemi windows dove ogni disco montato ha un suo percorso specifico, quindi vi è la possibilità di selezionare percorsi da diversi dischi mentre nei sistemi unix c'è solo un percorso sorgente, cioè "/", in quel caso l'apertura del popup non viene eseguita.

```
private void openRootSelect() {
    File[] rootsArray = File.listRoots();
    ArrayList<File> roots = new ArrayList<File>(Arrays.asList(rootsArray));
    if (roots.size() == 1) {
        root = roots.get(0);
        openFileSelect();
    } else {
        Grid<File> rootsGrid = new Grid<File>();
        Dialog rootDialog = new Dialog();
        SelectionListener<Grid<File>, File> listener =
            new SelectionListener<Grid<File>, File>() {
            @Override
            public final void selectionChange(SelectionEvent<Grid<File>, File> event) {
                Optional<File> selected = event.getFirstSelectedItem();
                if (selected.isPresent()) {
                    root = selected.get();
                    rootDialog.close();
                    openFileSelect();
                }
            }
        };

        rootDialog.setCloseOnOutsideClick(false);
        rootDialog.add(new Label("Select root path"));
        rootsGrid.setItems(roots);
        rootsGrid.addColumn(File::getAbsolutePath).setHeader("Root");
        rootsGrid.addSelectionListener(listener);
        rootDialog.add(rootsGrid);
        rootsGrid.setVisible(true);
        rootDialog.open();
    }
}
```

5.3.4.2 Metodo openFileSelect

Il metodo openFileSelect apre un popup dove si può scegliere il percorso di un file o cartella da aggiungere al servizio deduplicator. I percorsi sono caricati dal root del filesystem scelto in precedenza dal popup aperto dal metodo *openRootSelect*.

```
private void openFileSelect() {
    FileSystemData rootData = new FileSystemData(root, false);
    FileSystemDataProvider fileSystem = new FileSystemDataProvider(rootData);

    TreeGrid<File> fileBrowser = new TreeGrid<>();

    fileBrowser.setDataProvider(fileSystem);
    fileBrowser.addSelectionListener(event -> {
        Optional<File> selected = event.getFirstSelectedItem();
        pathTextField.setValue(selected.get().getAbsolutePath());
    });

    fileBrowser.addHierarchyColumn(File::getAbsolutePath).setHeader("Path");

    Dialog dialog = new Dialog();
    Button confirmButton = new Button("Close", button -> {
        dialog.close();
    });

    VerticalLayout layout = new VerticalLayout();
    layout.add(new Label("Select file or folder"), fileBrowser, confirmButton);
    layout.setMinWidth("50em");
    layout.setAlignItems(Alignment.CENTER);
    dialog.setCloseOnOutsideClick(false);
    dialog.add(layout);
    dialog.open();
}
```

Per caricare i percorsi viene utilizzata la classe *FilesystemDataProvider* che offre il caricamento dei dati solo una volta che sono richiesti, inoltre si occupa di gestire tutte le richieste per l'ottenimento dei percorsi figli di una cartella. I dati vengono rappresentati tramite il componente *TreeGrid* di Vaadin con il quale si può rappresentare molto bene la struttura dei file sul disco.

Una volta scelto il file o la cartella desiderata il suo percorso viene scritto nel campo per il percorso. Utilizzano questo metodo si evita che l'utente inserisca un percorso non valido.

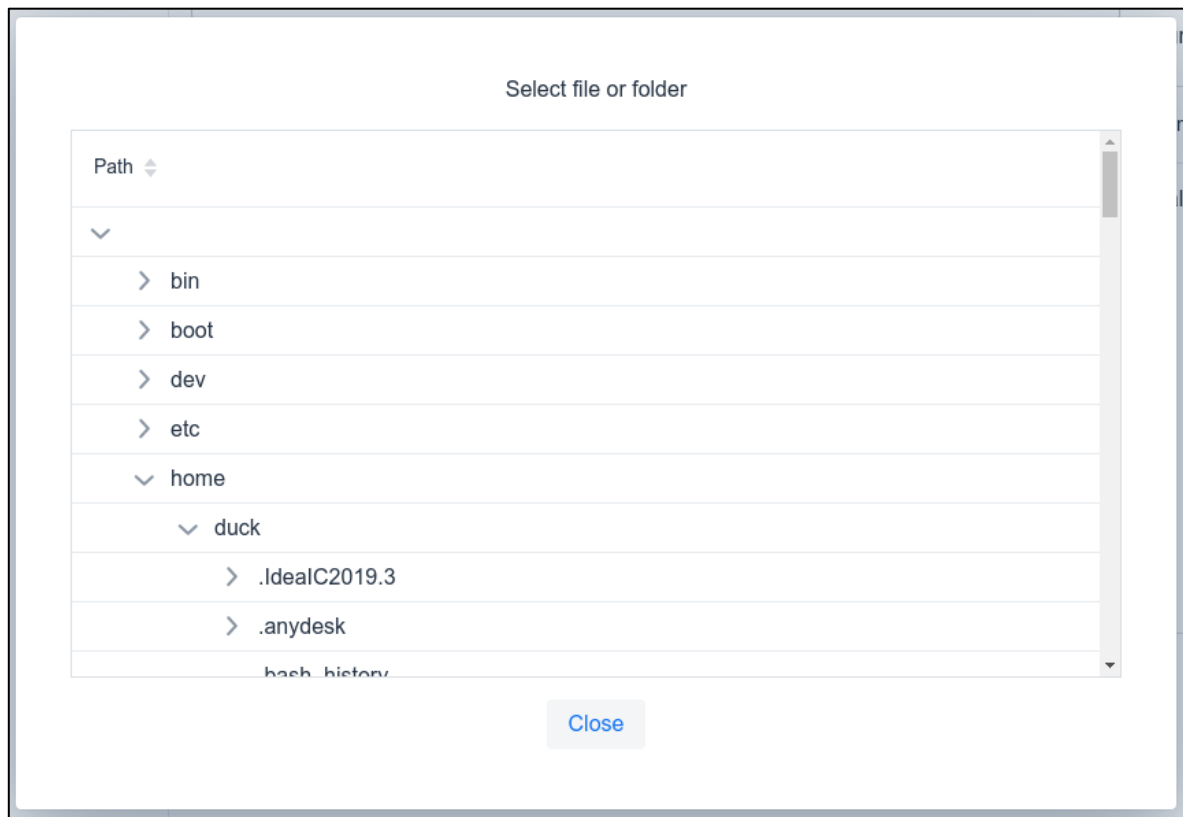


Figura 22 Popup per la selezione del percorso da aggiungere

5.3.4.3 Metodo updatePaths

Il metodo `updatePath` viene usato per aggiornare la tabella dei percorsi che sono presenti nella memoria del servizio deduplicator.

Il metodo `updatePath` viene invocato alla fine del costruttore o quando viene fatta una modifica sui percorsi.

Per prima vengono richiesti i percorsi sul server facendo una richiesta di tipo GET al percorso *path/*. Dopo che viene verificata la risposta del server, viene tentata una conversione dei dati ricevuti da *String* ad un array di *JSONObject*. La stringa ricevuta viene per prima trasformata in un oggetto di tipo *JSONArray* e poi trasformata in un array di *JSONObject* tramite il metodo *getArray* della classe *Utils*.

```

ResponseEntity<String> response = client.get("path/");

if (response != null && response.getStatusCode().equals(HttpStatus.OK)) {
    try {
        JSONObject[] array = Utils.getArray((JSONArray) parser.parse(response.getBody()));
        List<GlobalPath> paths = new ArrayList<GlobalPath>();

        for (JSONObject jsonObject : array) {
            try {
                paths.add(encoder.getMapper().readValue(jsonObject.toJSONString()
                    .replace("date", "lastModified"), GlobalPath.class));
            } catch (Exception e) {
                System.out.println(e.getMessage());
                Logger.getGlobal().log(Level.SEVERE, "An exception occurred while loading paths: " +
                    e.getMessage());
            }
        }

        pathGrid.setItems(paths);
    }
}

```

5.3.4.4 Metodo savePath

Il metodo *savePath* si occupa di mandare il percorso scelto dall'utente al servizio deduplicator.

Questo viene fatto usando il metodo *savePath* del Client.

```

ResponseEntity<String> response = client.savePath(pathTextField.getValue(), type);

```

Una volta ricevuta la risposta, essa viene analizzata e nel caso di errore viene mostrato un messaggio d'errore appropriato.

Il metodo *savePath* del client fa semplicemente una chiamata al metodo *put* del Client con i valori impostati usando una *MultiValueMap* che salva i dati in un formato tipo chiave-valore.

```

public ResponseEntity<String> savePath(String path, String type) {

    MultiValueMap<String, Object> values = new LinkedMultiValueMap<>();
    values.add("path", path);
    values.add("ignorePath", type.equals("ignore"));
    return put("path/", values);
}

```

I metodi *modifyPath* e *deletePath* funzionano allo stesso modo, con la differenza che cambia il metodo chiamato nel Client a post e delete rispettivamente.

5.3.5 ScanView

La ScanView si viene usata per gestire le scansioni offrendo all'utente la possibilità di iniziare una nuova scansione, mettere in pausa una scansione che è in esecuzione, riprendere una scansione che è stata messa in pausa e fermare una scansione. All'utente vengono mostrate le informazioni utili della scansione come: i file scansionati, i file totali, la stima del tempo rimanente, la data d'inizio e in basso alla schermata una barra di caricamento.

5.3.5.1 La thread dello stato

La thread che controlla lo stato della scansione viene inizializzata tramite il metodo *createStatusThread*. Quando viene fatta partire essa invoca il metodo *getStatus* del client che a sua volta fa una richiesta di tipo GET all'indirizzo *scan/status*. La *status thread* aggiorna i valori contenuti nei label della pagina grazie alla chiamata del metodo *updateStatus*. La chiamata viene fatta grazie al metodo *access* della classe *UI* che riceve accesso esclusivo alla pagina per il cambio dei valori e dopo esegue il codice presente nella classe *Command* che viene passata come parametro. Per apportare le modifiche viene fatta la chiamata al metodo *push* della classe *UI*.

5.3.5.1.1 Calcolo stima del tempo

Per il calcolo della stima del tempo viene usata la differenza nel numero dei file scansionati da una richiesta all'altra. Sapendo quanto tempo è passato da una richiesta all'altra grazie alla costante *POLLING_DELAY* possiamo calcolare quanti file sono stati scansionati nell'arco del tempo tra una richiesta e l'altra. Questo valore viene salvato nella variabile *pace* che viene usata subito dopo per calcolare quanto tempo serve per scansionare i file rimanenti.

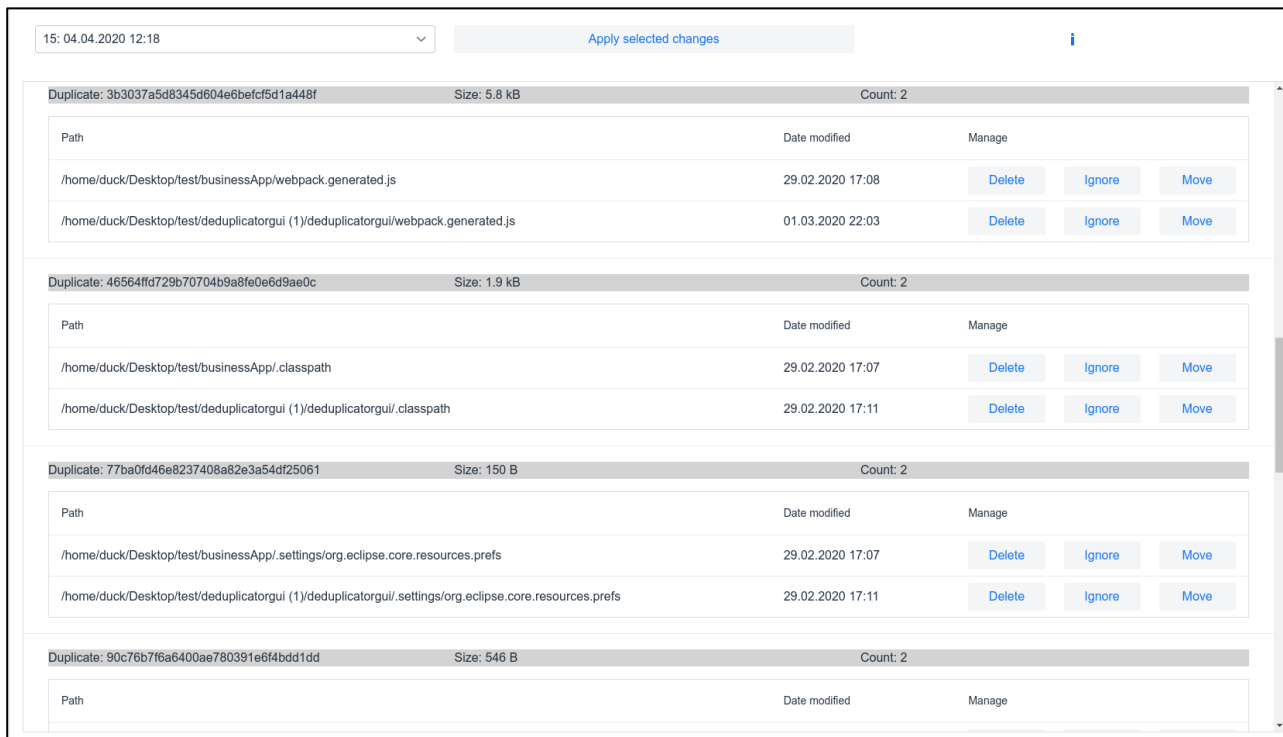
```
while (!isInterrupted() && scanProgress < 1f && scanProgress != -1f && scanProgress >= 0f) {
    synchronized (statusMonitor) {
        if (paused) {
            statusMonitor.wait();
        }
    }
    JSONObject response = client.getStatus();
    if (response != null) {
        if (response.get("message") == null) {
            int filesScanned = Integer.parseInt(response.get("fileCount").toString());
            int filesScannedDelta = filesScanned - filesScannedOld;
            int totFiles = Integer.parseInt(response.get("totalFiles").toString());
            float pace = ((float) POLLING_DELAY) / (float) filesScannedDelta;
            float timeLeft = ((totFiles - filesScanned) * pace) / 1000;

            Command command = (Command) () -> {
                updateStatus(true, totFiles, filesScanned, Long.parseLong(response.get("timestamp")
                    .toString()),
                    Float.parseFloat(response.get("progress").toString()), (int) timeLeft);
                ui.push();
            };

            scanProgress = Float.parseFloat(response.get("progress").toString());
            ui.access(command);
            filesScannedOld = filesScanned;
            synchronized (this) {
                this.wait(POLLING_DELAY);
            }
        }
    }
}
```

5.3.6 ReportView

La ReportView viene usata per visualizzare i rapporti delle scansioni che contengono i duplicati trovati. I file duplicati possono essere eliminati, ignorati o spostati in un'altra posizione.



Duplicate	Size	Count									
3b3037a5d8345d604e6befcf5d1a448f	5.8 kB	2									
<table border="1"> <thead> <tr> <th>Path</th> <th>Date modified</th> <th>Manage</th> </tr> </thead> <tbody> <tr> <td>/home/duck/Desktop/test/businessApp/webpack.generated.js</td> <td>29.02.2020 17:08</td> <td>Delete Ignore Move</td> </tr> <tr> <td>/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/webpack.generated.js</td> <td>01.03.2020 22:03</td> <td>Delete Ignore Move</td> </tr> </tbody> </table>			Path	Date modified	Manage	/home/duck/Desktop/test/businessApp/webpack.generated.js	29.02.2020 17:08	Delete Ignore Move	/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/webpack.generated.js	01.03.2020 22:03	Delete Ignore Move
Path	Date modified	Manage									
/home/duck/Desktop/test/businessApp/webpack.generated.js	29.02.2020 17:08	Delete Ignore Move									
/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/webpack.generated.js	01.03.2020 22:03	Delete Ignore Move									
46564ffd729b70704b9a8fe0e6d9ae0c	1.9 kB	2									
<table border="1"> <thead> <tr> <th>Path</th> <th>Date modified</th> <th>Manage</th> </tr> </thead> <tbody> <tr> <td>/home/duck/Desktop/test/businessApp/.classpath</td> <td>29.02.2020 17:07</td> <td>Delete Ignore Move</td> </tr> <tr> <td>/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.classpath</td> <td>29.02.2020 17:11</td> <td>Delete Ignore Move</td> </tr> </tbody> </table>			Path	Date modified	Manage	/home/duck/Desktop/test/businessApp/.classpath	29.02.2020 17:07	Delete Ignore Move	/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.classpath	29.02.2020 17:11	Delete Ignore Move
Path	Date modified	Manage									
/home/duck/Desktop/test/businessApp/.classpath	29.02.2020 17:07	Delete Ignore Move									
/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.classpath	29.02.2020 17:11	Delete Ignore Move									
77ba0fd46e8237408a82e3a54df25061	150 B	2									
<table border="1"> <thead> <tr> <th>Path</th> <th>Date modified</th> <th>Manage</th> </tr> </thead> <tbody> <tr> <td>/home/duck/Desktop/test/businessApp/.settings/org.eclipse.core.resources.prefs</td> <td>29.02.2020 17:07</td> <td>Delete Ignore Move</td> </tr> <tr> <td>/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.settings/org.eclipse.core.resources.prefs</td> <td>29.02.2020 17:11</td> <td>Delete Ignore Move</td> </tr> </tbody> </table>			Path	Date modified	Manage	/home/duck/Desktop/test/businessApp/.settings/org.eclipse.core.resources.prefs	29.02.2020 17:07	Delete Ignore Move	/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.settings/org.eclipse.core.resources.prefs	29.02.2020 17:11	Delete Ignore Move
Path	Date modified	Manage									
/home/duck/Desktop/test/businessApp/.settings/org.eclipse.core.resources.prefs	29.02.2020 17:07	Delete Ignore Move									
/home/duck/Desktop/test/deduplicatorgui (1)/deduplicatorgui/.settings/org.eclipse.core.resources.prefs	29.02.2020 17:11	Delete Ignore Move									
90c76b7f6a6400ae780391e6f4bdd1dd	546 B	2									
<table border="1"> <thead> <tr> <th>Path</th> <th>Date modified</th> <th>Manage</th> </tr> </thead> <tbody> </tbody> </table>			Path	Date modified	Manage						
Path	Date modified	Manage									

Figura 23 Visuale generale della ReportView

5.3.6.1 Metodo initiateGrid

Il metodo *initiateGrid* viene invocato quando l'utente sceglie un rapporto dal menu a tendina in alto a sinistra contenente tutti i rapporti. Una volta scelto un rapporto vengono subito caricati i file duplicati trovati in quel rapporto

```
private void initiateGrid(String reportSelectValue) {
    duplicatesGrid.removeAllColumns();
    String reportId = forMainView ? reportSelectValue : reportSelect.getValue().split(":")[0];
    if (reportId != null) {
        if (!reportId.isBlank()) {
            duplicateGridService = new DuplicateGridService(reportId, client, forMainView);
            if (duplicateGridService.getTotalDuplicatesCount() > 0) {
                DataProvider<DuplicateGrid, Void> dataProvider = DataProvider
                    .fromCallbacks(query -> {
                        int offset = query.getOffset();
                        int limit = query.getLimit();
                        List<DuplicateGrid> duplicates = getDuplicateGridService()
                            .fetchInsideGrids(offset, limit);
                        return duplicates.stream();
                    }, query -> duplicateGridService.getTotalDuplicatesCount());
                duplicatesGrid.setDataProvider(dataProvider);
            }
        }
    }
}
```


5.3.6.2 Metodo getReportInfo

Il metodo getReportInfo ricava le informazioni sul rapporto scelto e le rappresenta in un popup.

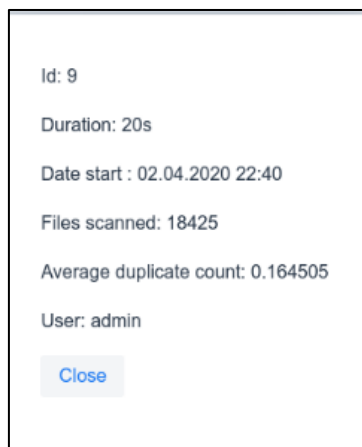


Figura 24 Popup contenente le informazioni su un rapporto

```
private void getReportInfo() {
    Dialog dialog = new Dialog();
    VerticalLayout verticalLayout = new VerticalLayout();
    Report report = client.getReport(reportSelect.getValue().split(":")[0]);
    if (report != null) {
        Calendar dateStartCalendar = Calendar.getInstance();
        dateStartCalendar.setTimeInMillis(report.getStart());
        Label durationLabel = new Label("Duration: " + report.getDuration() / 1000 + "s");
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy HH:mm");
        Label dateStartLabel = new Label("Date start : " + dateFormat.format(dateStartCalendar.
getTime()));
        Label filesScannedLabel = new Label("Files scanned: " + report.getFilesScanned().toStri
ng());
        Label averageDuplicateCountLabel = new Label(
            "Average duplicate count: " + report.getAverageDuplicateCount().toString());
        Label userLabel;
        try {
            userLabel = new Label("User: " + ((JSONObject) parser.parse(report.getUser()))
                .get("username"));
        } catch (ParseException pe) {
            Logger.getGlobal().log(Level.WARNING, "Unable to parse user of report");
            userLabel = new Label("User: unknown");
        }
        Label idLabel = new Label("Id: " + report.getId().toString());

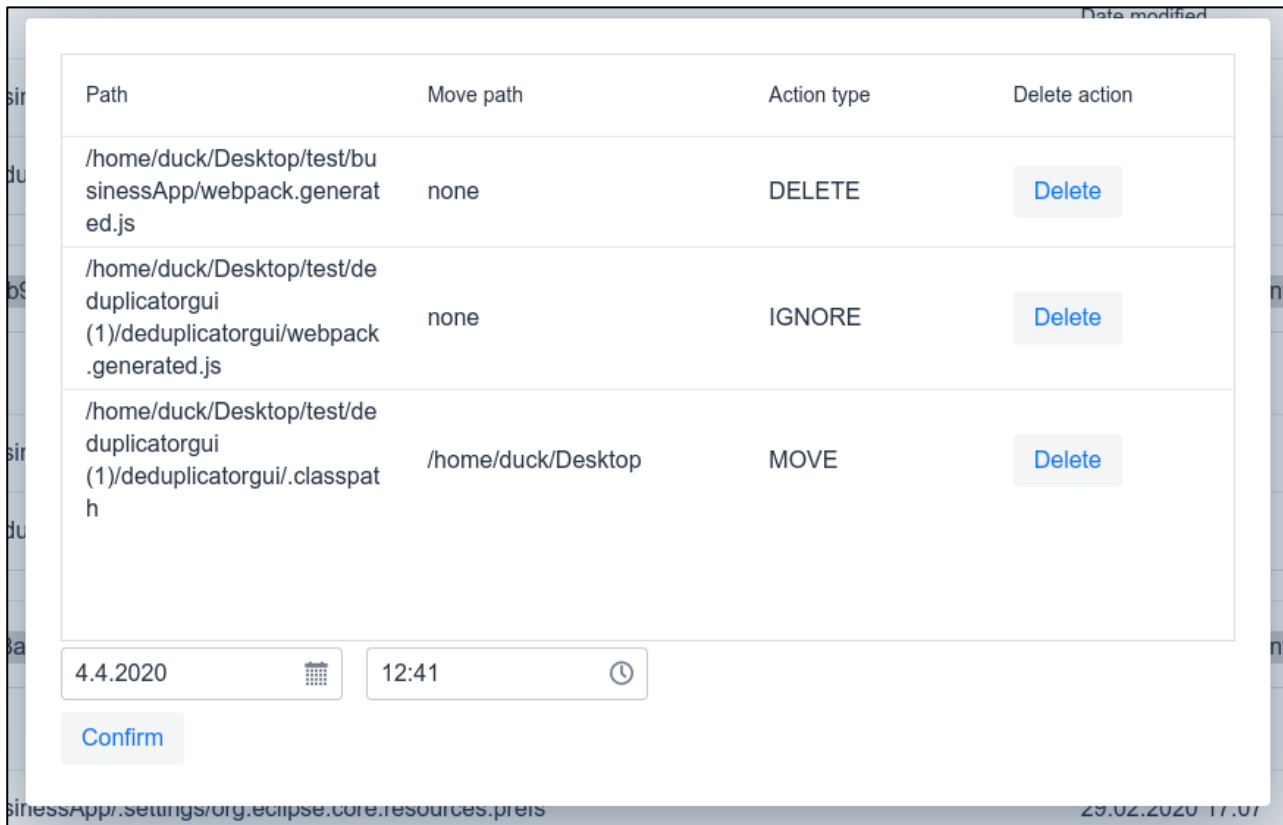
        verticalLayout.add(idLabel, durationLabel, dateStartLabel, filesScannedLabel,
            averageDuplicateCountLabel, userLabel, new Button("Close", event -> dialog.close()));

        dialog.add(verticalLayout);
        dialog.open();
    } else {
        Notification.show("Unable to get report information", settings.getNotificationLength(),
            Position.TOP_END)
            .addThemeVariants(NotificationVariant.LUMO_ERROR);
        Logger.getGlobal().log(Level.SEVERE, "Unable to get report information");
    }
}
```



5.3.6.3 Metodo checkActions

La operazione da eseguire su un file si sceglie cliccando uno dei 3 bottoni della colonna *Manage* della tabella interna.

Schiacciando sul tasto *Apply selected changes* in altro centrale della finestra viene fatta una chiamata al metodo *checkActions* che controlla tutti i percorsi contenuti nelle tabelle e popup che riassume tutte le operazioni da svolgere. Nel popup riassuntivo si può impostare la data di esecuzione delle modifiche.



Path	Move path	Action type	Delete action
/home/duck/Desktop/test/businessApp/webpack.generated.js	none	DELETE	Delete
/home/duck/Desktop/test/deduplicatorgui(1)/deduplicatorgui/webpack.generated.js	none	IGNORE	Delete
/home/duck/Desktop/test/deduplicatorgui(1)/deduplicatorgui/.classpath	/home/duck/Desktop	MOVE	Delete

4.4.2020  12:41 

[Confirm](#)

Figura 25 Il popup riassuntivo

Con il seguente codice vengono raccolti tutti i percorsi caricati da parte del *duplicateGridService* e poi filtrati grazie al metodo *removeIf* della classe *List* di java. Nel caso che l'azione di un file è di tipo *MOVE* viene anche indicato il percorso dove verrà spostato il file.

```
private void checkActions() {
    Logger.getGlobal().log(Level.INFO, "Check actions dialog opened");
    if (duplicateGridService != null) {
        List<GlobalPath> paths = new ArrayList<GlobalPath>(duplicateGridService.getPaths());
        Grid<GlobalPath> actionsGrid = new Grid<GlobalPath>();
        paths.removeIf(globalPath -> globalPath.getAction().getType() == null);
        actionsGrid.setItems(paths);
        actionsGrid.addColumn(GlobalPath::getPath).setHeader("Path").setFlexGrow(2);
        actionsGrid.addColumn(path -> {
            if (path.getAction().getType().equals(ActionType.MOVE)) {
                return path.getAction().getNewPath();
            } else {
                return "none";
            }
        }).setHeader("Move path").setFlexGrow(2);
        actionsGrid.addColumn(path ->
            path.getAction().getType()).setHeader("Action type").setFlexGrow(1);
        actionsGrid.addColumn(new ComponentRenderer<Button, GlobalPath>(path ->
            new Button("Delete", event -> {
                paths.remove(path);
                actionsGrid.getDataProvider().refreshAll();
            }))).setHeader("Delete action").setFlexGrow(1);
        actionsGrid.setClassName("inside-grid");
    }
}
```

Quando viene fatta di nuovo la conferma sulle azioni da eseguire viene chiamato il metodo *addActions* della classe *Client*.

```
Button applyDialogConfirmButton = new Button("Confirm", event -> {
    Logger.getGlobal().log(Level.INFO, "Actions confirmed");
    client.addActions(LocalDateTime.of(datePicker.getValue(), timePicker.getValue()), paths,
        null);
    applyDialog.close();
});
```

Il metodo `addActions` della classe `Client` crea un nuovo scheduler con la data impostata nel popup e aggiunge tutte le azioni da eseguire.

```
public HttpStatus addActions(LocalDateTime time, List<GlobalPath> actions,
                             String schedulerId) {
    if (schedulerId == null) {
        ResponseEntity<String> response = insertSchedule(time, null, null, "One off");
    }
    ...
}
```

```
for (GlobalPath path : actions) {
    Action action = path.getAction();
    ResponseEntity<String> response = insertAction(action.getType(), path.getPath(), action.getNewPath(),
    schedulerId);
    if (response != null) {
        Logger.getGlobal().log(Level.INFO, "Added action " + path.getPath() + " status: " + response.getStatusCode());
        if (response.getStatusCode() == HttpStatus.INTERNAL_SERVER_ERROR) {
            Notification.show("Unable to add action of: " + path.getPath(), settings.getNotificationLength(),
            Notification.Position.TOP_END).addThemeVariants(NotificationVariant.LUMO_ERROR);
            Logger.getGlobal().log(Level.WARNING, "Unable to add action of: " + path.getPath());
        }
    } else {
        Logger.getGlobal().log(Level.SEVERE, "No response from server - unable to add action of " + path.getPath());
    }
}
}
```

5.3.7 ScheduleView

La `ScheduleView` offre all'utente la possibilità di impostare delle scansioni pianificate.

Si possono pianificare scansioni singole (One off), scansioni giornaliere (Daily), scansioni settimanali (Weekly) e scansioni mensili (Monthly).

Figura 26 Visuale generale della `ScheduleView`

Start date

5.4.2020



Start time

15:34



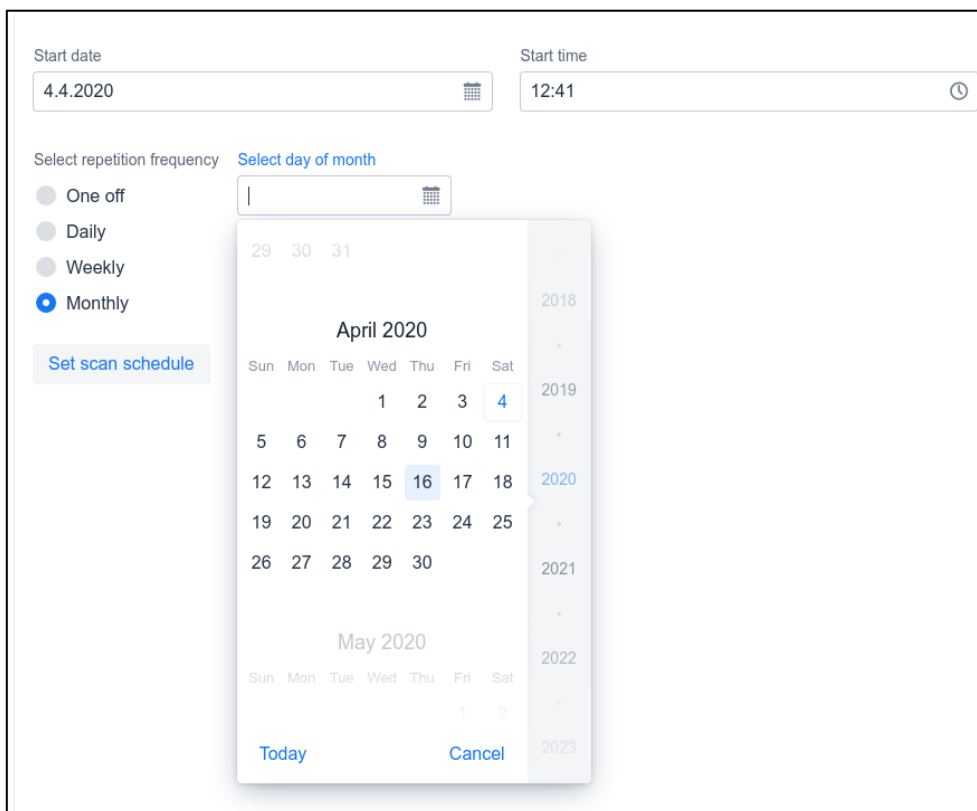
Select repetition frequency

- ☒ One off
- ☐ Daily
- ☐ Weekly
- ☐ Monthly

Set scan schedule

Delete a schedule

Figura 27 Visuale schermata scheduler con la scelta di scansione settimanale



The screenshot shows the scheduler interface with the following elements:

- Start date:** 4.4.2020
- Start time:** 12:41
- Select repetition frequency:**
 - ☐ One off
 - ☐ Daily
 - ☐ Weekly
 - ☒ Monthly
- Select day of month:** A dropdown menu is open, showing a calendar for April 2020. The date 16 is selected.
- Buttons:** "Set scan schedule" and "Cancel" are visible at the bottom of the calendar overlay.

Figura 28 Visuale scheduler con la scelta di scansione mensile

Prima della di tentare di inserire lo scheduler nel servizio viene fatto un controllo che l'ora e data scelta siano in futuro grazie al metodo *checkDateTimeValues*.

```
Button button = new Button("Set scan schedule", event -> {
    if (checkDateTimeValues(LocalDate.of(datePicker.getValue(), timePicker.getValue()))){
        ResponseEntity<String> response = client.insertScheduledScan(LocalDate.of(
            datePicker.getValue(), timePicker.getValue()), (weekNumber != null ? weekNumber : "")
            .toString(), (monthNumber != null ? monthNumber : "").toString(), radioButtonGroup.getValue
            ());
    }
});
```

Si possono anche eliminare dei schedule impostato cliccando sul bottone delete schedule che apre un popup con tutti i scheduler attualmente attivi, cliccando su una riga si apre un altro popup con al richiesta della conferma dell'eliminazione dello scheduler.

5.3.8 DashboardView

La DashboardView consente all'utente di cambiare delle impostazioni della applicazione. Le impostazioni che si possono cambiare sono: la password dell'utente, l'intervallo con cui vengono aggiornati i dati nella ScanView, la durata delle notifiche, la posizione dei file di log e nel caso che l'utente non sia l'admin, anche l'username. Inoltre, si può anche scaricare e cancellare l'ultimo file di log e aggiungere un utente.

Le modifiche apportate con i selettori sul lato destro della schermata sono immediatamente applicate.

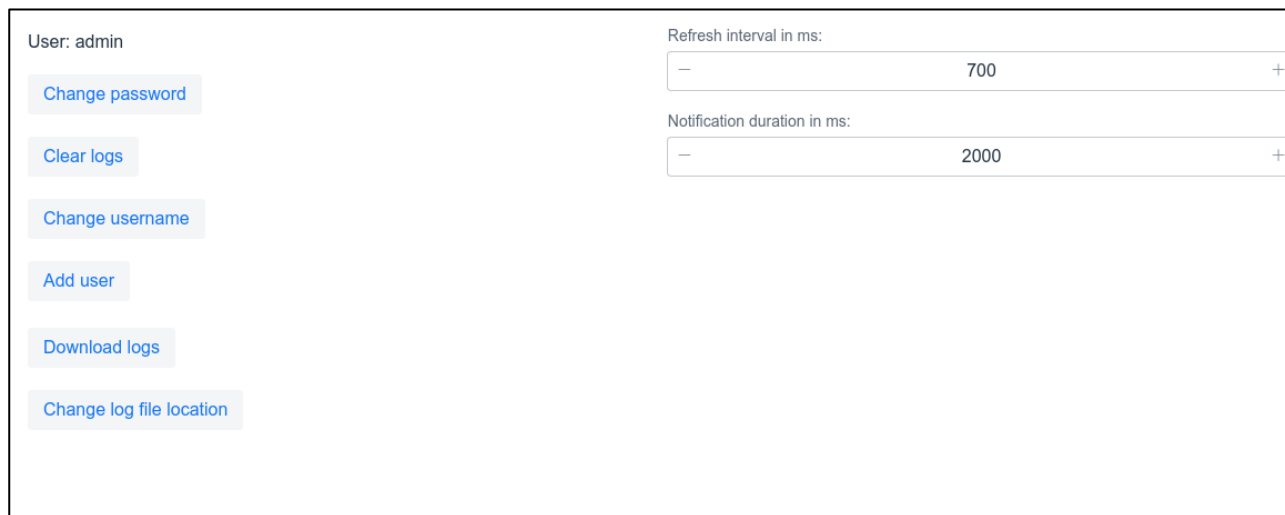
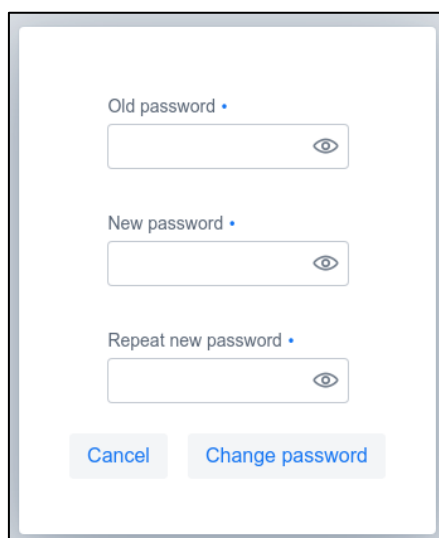


Figura 29 Visuale generale della DashboardView



Old password •

New password •

Repeat new password •

Cancel Change password

Figura 30 Popup del cambio della password

5.3.9 Script di installazione

5.4 Configurazione

5.5 Logger

Il log dell'applicazione vengono svolti utilizzando la classe *Logger* presente in java. Essa offre la possibilità di definire il livello di un messaggio e di impostare una formattazione per la scrittura dei log su un file.

Usando il metodo *startup* nella classe principale *Application* viene chiamato il metodo *setup* della classe *MyLogger* che imposta la formattazione e dove verranno salvati i log. La classe *MyLogger* si ottiene utilizzando la classe *ApplicationContext* e l'annotazione *@Component*, utilizzata nella classe *MyLogger*, di **Spring boot**

```
@Autowired
ApplicationContext context;

@PostConstruct
public void startup() {
    try {
        MyLogger logger = (MyLogger) context.getBean("myLogger");
        logger.setup();
    } catch (IOException ioe) {
        System.out.println("Unable to setup logger");
        ioe.printStackTrace();
    }
}
```

Il metodo *setup* prende il percorso dove verrà salvato il file di log dalla classe *Settings* che lo legge del file di configurazione. Tutte le impostazioni vengono applicate al logger Globale che poi viene utilizzato in tutte le altre classi.

```
public void setup() throws IOException {
    String logPath = settings.getLogPath();
    Logger logger = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
    Path p = Paths.get(logPath);
    File logFolder = new File(p.toAbsolutePath().toString());
    if (!logFolder.exists()) {
        if (Files.isWritable(Paths.get(logFolder.getParent()))) {
            if (!logFolder.mkdir()) {
                logPath = "";
            } else {
                logger.getGlobal().log(Level.SEVERE, "Unable to make log directory");
                logger.getGlobal().log(Level.INFO, "Writing logs to current working directory");
            }
        } else {
            logger.getGlobal().log(Level.SEVERE, "Unable to write to parent directory");
            logger.getGlobal().log(Level.INFO, "Writing logs to current working directory");
        }
    }
    FileHandler fileHandler = new FileHandler(logPath + "log." + System.currentTimeMillis() + ".txt");
    fileHandler.setFormatter(new LogFormatter());
    logger.addHandler(fileHandler);
}
```


Lo stile dei log che verranno scritti sul file di log è definito dalla classe *LogFormatter* che estende la classe *Formatter* di java. Ogni riga del file di log contiene: il livello del log all'inizio della riga, separato da un tabulatore la data e orario dell'avvenimento del log, separato da uno spazio segue il messaggio del log.

I log che hanno il livello impostato su *SEVERE* e *WARNING* includono anche, dopo il messaggio, il metodo e la classe nel quale si è verificato l'errore.

```
public class LogFormatter extends Formatter {

    @Override
    public String format(LogRecord record) {
        StringBuilder builder = new StringBuilder();
        builder.append("[").append(record.getLevel().getName()).append("]").append("\t");
        if (record.getLevel().equals(Level.INFO)) {
            builder.append("\t");
        }
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSSS");
        Calendar cal = Calendar.getInstance();
        cal.setTimeInMillis(record.getMillis());
        builder.append(sdf.format(cal.getTime())).append(" ");

        builder.append(record.getMessage());
        if (record.getLevel().equals(Level.SEVERE) || record.getLevel().equals(Level.WARNING)){
            builder.append(" at ").append(record.getSourceMethodName()).append(" in ")
                .append(record.getSourceClassName());
        }
        builder.append("\n");
        return builder.toString();
    }
}
```

5.6 Gestione Autenticazione

La gestione dell'autenticazione è implementata facendo spunto al progetto di esempio messo a disposizione dagli sviluppatori di Vaadin.

La soluzione implementata da loro utilizza una classe creata apposta per il controllo dell'autenticazione. Nella mia implementazione quella classe si chiama *DeduplicatorGUIInitListener* e implementa l'interfaccia *VaadinServiceInitListener* che sovrascrive il metodo *serviceInit*.

```
public void serviceInit(ServiceInitEvent initEvent)
```

Il metodo *serviceInit* viene chiamato ogni volta che viene caricata una View, in questo metodo viene eseguito il controllo se l'utente ha fatto l'accesso. Sé l'utente non è autenticato, esso viene reindirizzato verso la LoginView.

Per indicare a Vaadin la classe che implementa l'interfaccia *VaadinServiceInitListener* è stato creato il file *com.vaadin.flow.server.VaadinServiceInitListener* nella cartella *resources/META-INF/services/* contenente il nome canonico della classe.

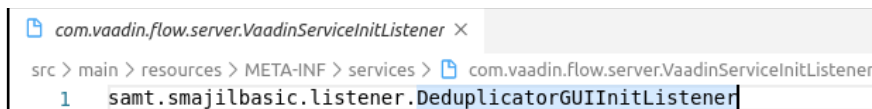


Figura 31 File che indica la classe che implementa il *VaadinServiceInitListener*

Il metodo *serviceInit* della classe *DeduplicatorGUIInitListener* contiene il seguente codice che grazie alla classe *AccessControlInterface* controlla se l'utente ha eseguito l'accesso nell'attuale sessione:

```
@Override
public void serviceInit(ServiceInitEvent initEvent) {
    final AccessControlInterface accessControl = AccessControlFactory.getInstance()
        .createAccessControl();
    initEvent.getSource().addUIInitListener(uiInitEvent -> {
        uiInitEvent.getUI().addBeforeEnterListener(event -> {
            if (!accessControl.isUserSignedIn() && !LoginView.class
                .equals(event.getNavigationTarget()))
                event.rerouteTo(LoginView.class);
        });
    });
}
```

Per l'effettivo controllo se l'utente è autenticato viene usato il metodo *isUserSignedIn* della classe *AccessControl* che controlla la presenza dell'utente nella memoria della sessione dell'applicazione.

```
public boolean isUserSignedIn() {
    return !CurrentUser.get().isEmpty();
}
```

Se il metodo *get* della classe *CurrentUser* ritorna *null*, questo sta a indicare che nella memoria dell'applicazione non è presente il nome dell'utente, quindi non è stato eseguito l'accesso perciò l'utente viene reindirizzato alla pagina di Login.

```
public static String get() {
    String currentUser = (String) UI.getCurrent().getSession()
        .getAttribute(Resources.CURRENT_USER_SESSION_ATTRIBUTE_KEY);
    if (currentUser == null) {
        return "";
    } else {
        return currentUser;
    }
}
```

6 Test

I test fatti sulle modifiche del progetto vecchio sono stati svolti con Postman 7.17.0.
Tutti i test rimanenti sono stati fatti con il browser Google Chrome 80.0.3987.149

6.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.


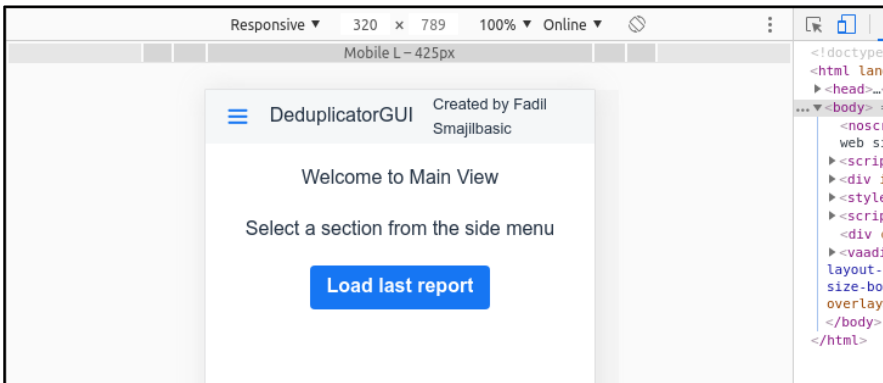
Test Case:	TC-01	Nome:	Verifica modifiche applicate.
Riferimento:	REQ-07 (01)		
Descrizione:	Verifica del giusto funzionamento della scansione dopo le modifiche applicate.		
Prerequisiti:	Il servizio deduplicator deve essere attivo. Il percorso della cartella test deve essere inserita nel database del servizio. L'utente deve sapere il username e password del servizio per poter fare la richiesta.		
Procedura:	<p>Ho creato la seguente struttura di cartelle e file per eseguire i test:</p> <pre> test/ ├── [15] asd.txt ├── [4.0K] cartella2 │ ├── [4.0K] cartella3 │ │ ├── [15] another_name.txt │ │ └── [9] un_altro_file.txt │ └── [0] file1.txt ├── [4.0K] folder1 │ └── [15] asd.txt └── [4.0K] test1 </pre> <p>I file <i>another_name.txt</i> e <i>asd.txt</i> sono uguali mentre i file <i>un_altro_file.txt</i> e <i>file1.txt</i> sono diversi.</p> <ul style="list-style-type: none"> ● Avviare la scansione ● Aspettare che la scansione finisce ● Fare richiesta di tipo GET sul seguente percorso: <ip servizio>:<porta servizio>/report/<id report> ● Analizzare i file trovati ● Fare richiesta di tipo GET sul seguente percorso: <ip servizio>:<porta servizio>/report/duplicate/<id report> ● Analizzare i duplicati trovati ● Fare richiesta di tipo GET sul seguente percorso: <ip servizio>:<porta servizio>/report/duplicate/<id report>/<hash del duplicato> 		

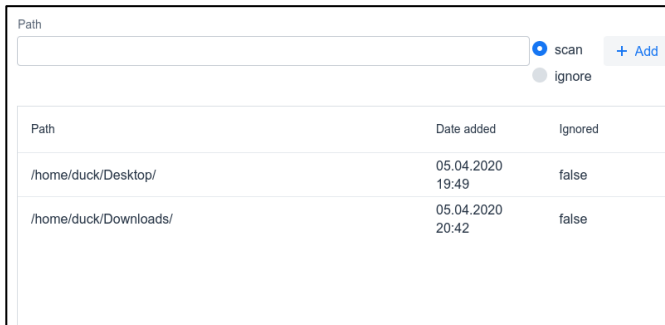
	<ul style="list-style-type: none"> Analizzare i file dei duplicati e verificare che ci sono solo i file <i>asd.txt</i> e <i>another_name.txt</i>
Risultati attesi:	<p>La scansione viene fatta partire, durante la scansione l'utente vede il progresso della scansione su terminale. Dopo che la scansione finisce viene salvato il rapporto nella tabella report del database e all'utente viene mandata indietro la risposta contenente il rapporto.</p> <p>Quando viene fatta la richiesta per trovare tutti i file del rapporto si attendono di trovare tutti i file della cartella test.</p> <p>Quando viene fatta la richiesta dei duplicati si attende di trovare un duplicato di grandezza 15 e numero di file 3</p> <p>Quando viene fatta la richiesta per ricavare i file del duplicato si attendono di vedere i 3 file che sono duplicati</p>


Test Case:	TC-02	Nome:	Verifica modifiche applicate.
Riferimento:	REQ-07 (03)		
Descrizione:	Verifica dello stato della scansione.		
Prerequisiti:	<p>Il servizio deduplicator deve essere attivo.</p> <p>Nel database ci deve essere il percorso di una cartella con più di 2 mila file.</p> <p>L'utente deve sapere il username e password del servizio per poter fare la richiesta.</p>		
Procedura:	<ul style="list-style-type: none"> Avviare la scansione Guardare l'output della scansione sul terminale Durante la scansione fare una richiesta di tipo GET sul seguente percorso: <ip servizio>:<porta servizio>/scan/status 		
Risultati attesi:	Si aspetta di ricevere una risposta che contiene il numero di file scansionati fino a quel punto e lo stato della scansione		

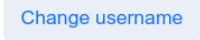
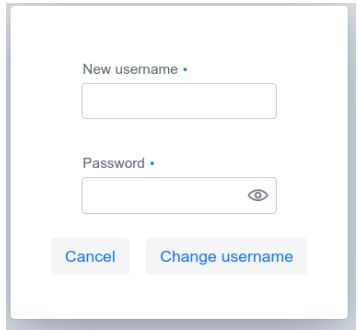
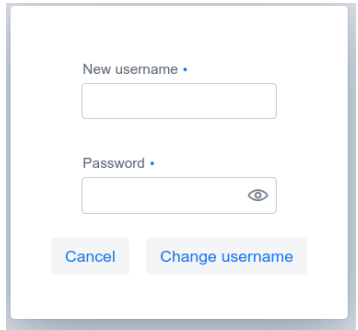
Test Case:	TC-03	Nome:	Verifica modifiche applicate.
Riferimento:	REQ-07 (02)		
Descrizione:	Verifica dell'orario dello scheduler.		
Prerequisiti:	<p>Il servizio deduplicator deve essere attivo.</p> <p>L'utente deve sapere il username e password del servizio per poter fare la richiesta.</p>		
Procedura:	<ul style="list-style-type: none"> Fare una richiesta di tipo PUT impostare il giorno del mese o giorno della settimana tramite i parametri della richiesta, inoltre impostare il flag <i>repeated</i> se si vuole che lo scheduler viene ripetuto e impostare il <i>timestamp</i> della data di avvio. Guardare l'output del servizio sul terminale o guardare il file di log del servizio per leggere il tempo che ha lo scheduler. Confrontare la data e orario mandati e quelli elaborati dal servizio 		
Risultati attesi:	Le due date e orari sono uguali.		


Test Case:	TC-04	Nome:	GUI moderna
Riferimento:	REQ-01		
Descrizione:	La GUI deve essere moderna e ispirata al material design		
Prerequisiti:	-		
Procedura:	-		
Risultati attesi:	Una GUI moderna e facile da usare		


Test Case:	TC-05	Nome:	Layout responsive
Riferimento:	REQ-02		
Descrizione:	La GUI si deve adattare a diverse grandezze dello schermo a dipendenza del dispositivo usato		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Installare il browser Google Chrome		
Procedura:	<p>Aprire il sito web con il browser Google Chrome</p> <p>Accedere al servizio con le credenziali impostate.</p> <p>Premere il tasto F12 sulla tastiera per aprire la finestra delle opzioni da sviluppatore</p> <p>Premere il bottone  sulla finestra per cambiare la visuale da quella desktop a quella mobile.</p> <p>Verificare che in ogni pagina si riescono a vedere i contenuti cambiano la grandezza dello schermo con la barra in alto alla finestra.</p>  <p>Figura 32 Visuale mobile dal browser Google Chrome</p> <p>Per verificare ulteriormente questo requisito occorre aprire il sito web tramite un telefono e/o un tablet.</p>		
Risultati attesi:	In ogni pagina si adatta alle diverse grandezze dello schermo..		

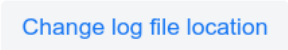
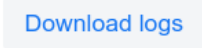
Test Case:	TC-06	Nome:	Ottenimento stato delle scansioni
Riferimento:	REQ-04		
Descrizione:	La GUI deve mostrare lo stato della scansione		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere al servizio tramite un browser		
Procedura:	<p>Aggiungere dei percorsi tramite la PathView con tanti file (almeno 20'000).</p>  <p>Figura 33 Percorsi con tanti file</p> <p>Avviare la scansione tramite la ScanView e prestare attenzione sulla barra di caricamento in alto e sulla stima di tempo.</p>		
Risultati attesi:	La barra di caricamento si riempie man mano che vengono scansionati più file e la stima del tempo rimane relativamente giusta durante tutta la durata della scansione.		

Test Case:	TC-07	Nome:	Pannello per la gestione del servizio: sotto requisito 01
Riferimento:	REQ-05 (01)		
Descrizione:	Test della funzione riguardo il cambio della password dell'utente.		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere al servizio tramite un browser		
Procedura:	<p>Aprire la DashboardView</p> <p>Cliccare sul bottone Change Password </p> <p>Nel popup che si apre inserire la vecchia password, la nuova password e ripetere la nuova password nei appositi campi.</p> <p>Confermare di nuovo il cambio della password cliccando sul bottone Change Password.</p> <p>Dopo che l'utente viene disconnesso, fare il login con la nuova password</p>		
Risultati attesi:	La password viene cambiata e l'utente può accedere con le nuove credenziali.		

Test Case:	TC-08	Nome:	Pannello per la gestione del servizio: sotto requisito 02
Riferimento:	REQ-05 (02)		
Descrizione:	Test della funzione riguardo cambio del username dell'utente.		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere, con un utente che non sia admin, al servizio tramite un browser		
Procedura:	<ol style="list-style-type: none"> 1. Aprire la DashboardView 2. Cliccare sul tasto Change  Username 3. Inserire il nuovo username e la password nel popup che si apre. <div data-bbox="798 680 1157 1008" data-label="Form">  </div> <ol style="list-style-type: none"> 4.  <p>Figura 34 Popup per l'inserimento dei dati per il cambio del username</p> <ol style="list-style-type: none"> 5. Aspettare che scade il timer e l'utente viene disconnesso. 6. Fare accesso con il nuovo username. 		
Risultati attesi:	L'utente può accedere con il nuovo username ma la password rimane uguale.		

Test Case:	TC-09	Nome:	Pannello per la gestione del servizio: sotto requisito 04
Riferimento:	REQ-05 (04)		
Descrizione:	Ci deve essere la possibilità di scaricare il file di log		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere al servizio tramite un browser		
Procedura:	<ol style="list-style-type: none"> 1. Aprire la DashboardView 2. Scaricare i log premendo il bottone Download logs  3. Il download del file parte. 		
Risultati attesi:	A dipendenza delle impostazioni del browser, viene aperto il dialogo del sistema per salvare il file dopodiché il file viene scaricato.		

Test Case:	TC-10	Nome:	Pannello per la gestione del servizio: sotto requisito 03
Riferimento:	REQ-05 (03)		
Descrizione:	Ci deve essere la possibilità di svuotare il file di log.		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere al servizio tramite un browser		
Procedura:	<ol style="list-style-type: none"> 1. Aprire la DashboardView 2. Scaricare i log premendo il bottone Download logs  3. Cliccare sul bottone Clear logs 4. Confermare di voler cancellare i log premendo di nuovo il bottone Clear logs nel popup che viene aperto. 5. Scaricare di nuovo il file di log. 6. Confrontare i due file e verificare che il nuovo file sia svuotato. 		
Risultati attesi:	Il secondo file scaricato deve avere solo la riga del log di livello INFO che è stato svuotato il file di log.		

Test Case:	TC-11	Nome:	Pannello per la gestione del servizio: sotto requisito 05
Riferimento:	REQ-05 (05)		
Descrizione:	Ci deve essere la possibilità di cambiare la posizione di salvataggio dei file di log		
Prerequisiti:	Avviare il servizio deduplicator Avviare il server della GUI Accedere al servizio tramite un browser Creare la nuova cartella dove verranno salvati i file di log.		
Procedura:	<ol style="list-style-type: none"> 1. Aprire la DashboardView 2. Cliccare sul bottone Change log file location  3. Scegliere la nuova cartella creata dove verranno salvati i log 4. Scaricare il file di log premendo il bottone  5. Verificare che ci sia la riga che indica che il file di log è stato spostato 6. Verificare, con un file browser (o via terminale), la presenza di un nuovo file di log nella posizione impostata 		
Risultati attesi:	Il nuovo file di log è stato creato sotto la nuova cartella e i log vengono scritti in modo regolare		

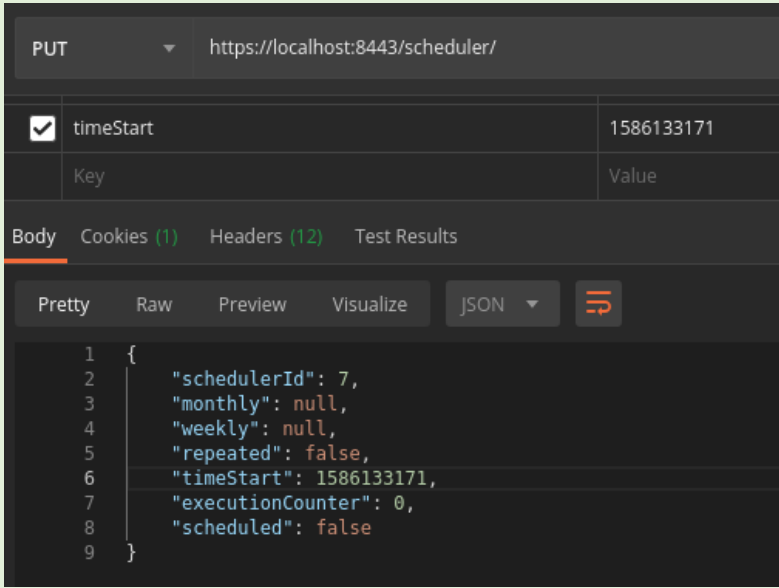
Test Case:	TC-012	Nome:	Architettura modulare
Riferimento:	REQ-006		
Descrizione:	Il progetto deve essere sviluppato in modo da poter aggiungere delle nuove funzionalità senza il bisogno di cambiare tutto il programma.		
Prerequisiti:	-		
Procedura:	-		
Risultati attesi:	Il progetto ha una struttura modulare.		

Test Case:	TC-013	Nome:	Avvio semplice della GUI
Riferimento:	REQ-008		
Descrizione:	La interfaccia utente deve essere semplice da avviare/eseguire.		
Prerequisiti:	-		
Procedura:	-		
Risultati attesi:	Un semplice metodo di avvio.		

6.2 Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l'errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l'errore con eventuali ipotesi di correzione.

Test case	Funzionamento	Commento
TC-01	OK	<p>Quando viene fatta la richiesta dei duplicati si ottiene un duplicato in formato json:</p> <pre>{ "hash": "<hash duplicato>", "size": 15, "count": 3 }</pre> <p>Quando viene fatta la richiesta per ricavare i file del duplicato si ottengono i 3 file che sono duplicati con tutte le loro informazioni.</p> <pre>[{ "path": "/home/duck/test/asd.txt", "lastModified": 1578521859224, "hash": "1d6155b60405bab055527913efd734a7", "size": 15 }, { "path": "/home/duck/test/cartella2/cartella3/another_name.txt", "lastModified": 1578521859224, "hash": "1d6155b60405bab055527913efd734a7", "size": 15 }, { "path": "/home/duck/test/folder1/asd.txt", "lastModified": 1578521859224, "hash": "1d6155b60405bab055527913efd734a7", "size": 15 }]</pre>
TC-02	OK	<p>Si ottiene lo stato della scansione nel seguente formato:</p> <pre>{ "fileCount": 2631, "progress": 0.6912503 }</pre>

Test case	Funzionamento	Commento
TC-03	OK	<p>Il tempo mandato e quello impostato è uguale</p>  <p>Figura 35 L'output ricevuto utilizzando Postman</p>
TC-04	OK	<p>La GUI è moderna e utilizza il tema Lumo con tutti i suoi componenti come questi indicati sul sito di Vaadin</p> <p>https://demo.vaadin.com/lumo-editor/</p>

TC-05

OK

Tutte le view si adattando alla grandezza dello schermo del dispositivo.

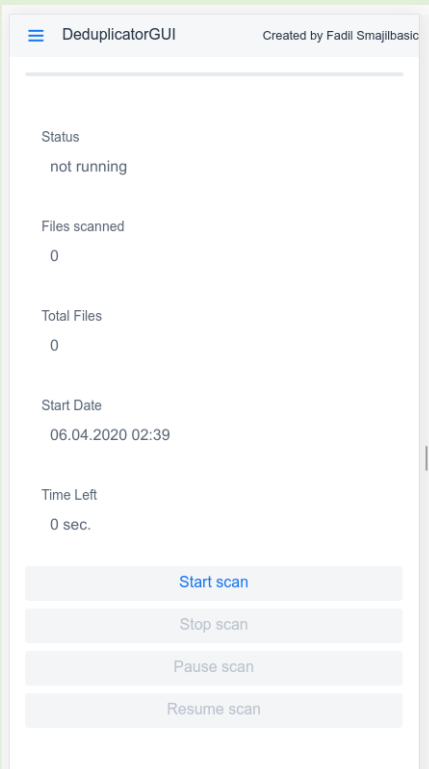


Figura 36 La ScanView in visuale da telefono

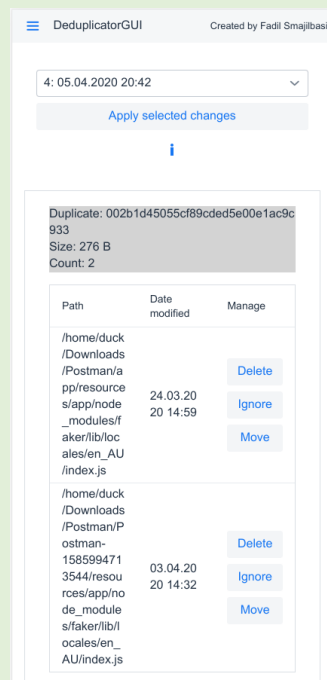
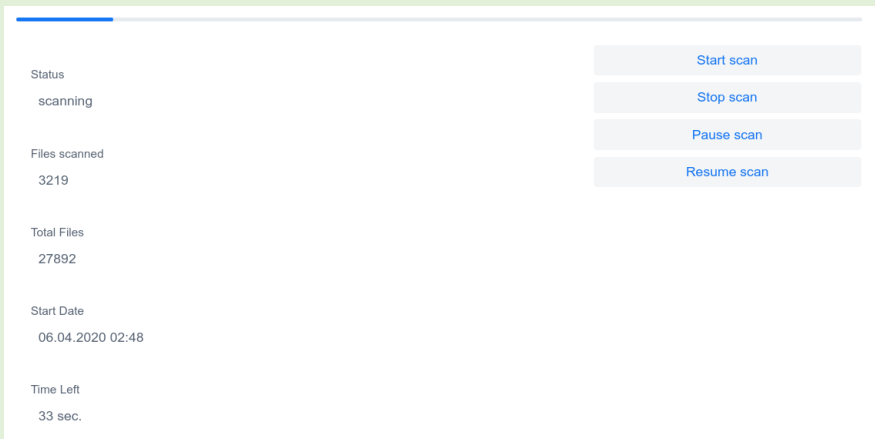
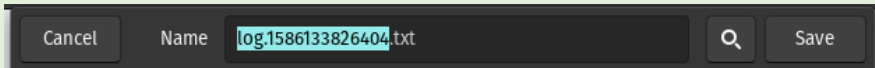
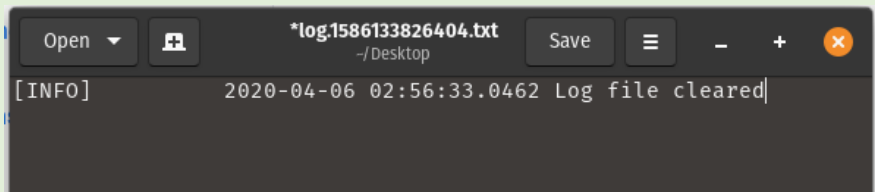
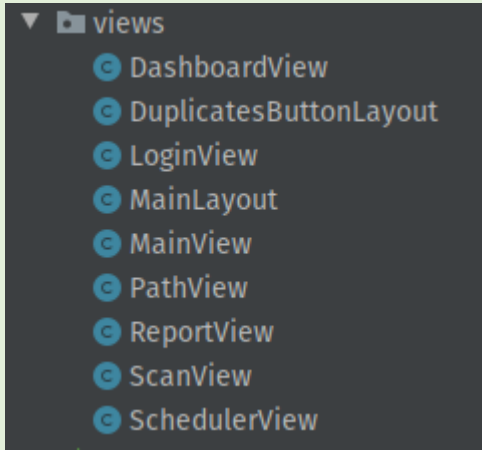


Figura 37 La ReportView in visuale da telefono

TC-06	OK	<p>Lo stato della scansione viene aggiornato in modo regolare secondo l'intervallo di tempo impostato nella DashboardView.</p>  <p>Figura 38 Una scansione in esecuzione</p>
TC-07	OK	La password viene cambiata e l'utente può accedere con le nuove credenziali.
TC-08	OK	L'utente può accedere con il nuovo username ma la password rimane uguale.
TC-09	OK	<p>Una volta premuto il bottone download logs, viene aperto il dialogo del sistema per selezionare la destinazione del file da scaricare, il nome del file è impostato automaticamente.</p>  <p>Figura 39 Il dialogo del sistema di Pop_Os per il salvataggio dei file</p>
TC-10	OK	<p>Dopo aver cancellato i log, il file scaricato viene aperto con gedit e si può vedere che contiene solo la riga <i>Log file cleared</i>.</p>  <p>Figura 40 Il log file scaricato</p>
TC-11	OK	<p>Dopo aver spostato il file di log, con l'applicazione tree si può vedere che nella cartella <i>new_logs</i> è stato creato il nuovo file <i>.txt</i> e un altro <i>.lck</i> che è il file di lock del handler.</p> <pre> 0 directories, 121 files duck@airpods-pro:~/projects/updated/deduplicator/Code/deduplicatorVaadinGUI\$ tree new_logs/ log/ new_logs/ ├── log.1586134890737.txt ├── log.1586134890737.txt.lck log/ ├── log.1586133826404.txt </pre> <p>Figura 41 output del comando tree</p>

TC-12	OK	<p>Il progetto è fatto in modo modulare perciò permette una semplice di diverse funzionalità. Basta aggiungere una nuova classe che fa da View, aggiungere il percorso nel menu di lato ed è tutto.</p>  <p>Figura 42 Struttura delle le view nel progetto</p>
TC-13	OK	TODO

6.3 Mancanze/limitazioni conosciute

Quando viene spostato un log, viene creato un nuovo file nella nuova posizione scelta che non contiene i contenuti del vecchio file di log, perciò i dati del file di log rimangono all'ultima posizione.

7 Consuntivo

Le ore alla fine del progetto ammontano a circa 148 ore, questo è per via dell'annullamento della gita che aggiunge una settimana di tempo al progetto.

Le view non sono state sviluppate tutte in un colpo, ma ci sono state delle varie modifiche e iterazioni durante tutta la fase di implementazione. Le ore di sviluppo di ogni view rappresentano in totale quanto tempo è stato dedicato.

Il diagramma di Gantt consuntivo è presente tra gli allegati del documento.

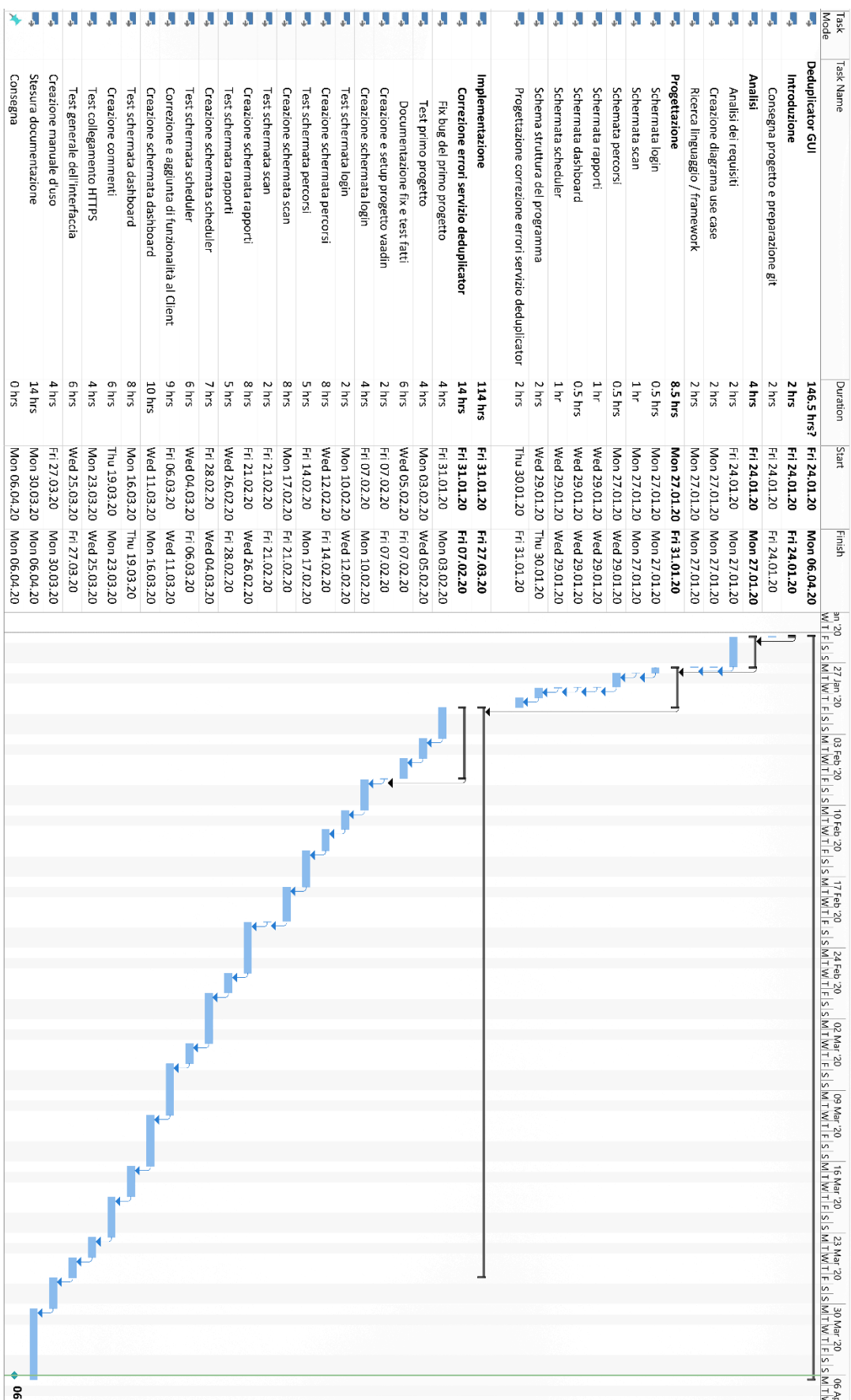


Figura 43 Gantt Consuntivo

8 Conclusioni

La nuova GUI sviluppata porta un cambiamento radicale alla GUI precedente, inoltre completa le sue funzionalità. Il servizio deduplicator adesso è semplice da usare e gestire.

8.1 Sviluppi futuri

Mettere assieme i due progetti in uno unico con le parti di backend e frontend.

Una volta messi assieme, bisognerebbe implementare una vera gestione degli utenti.

Mettendo assieme i due progetti si potrebbero anche fare vedere degli errori più specifici riguardo eventuali errori.

8.2 Considerazioni personali

In questo progetto ho imparato come usare il framework Vaadin e come creare una GUI user-friendly usando solo java.

9 Bibliografia

9.1 Sitografia

http://www.treccani.it/enciclopedia/repository_%28Lessico-del-XXI-Secolo%29/, repository in "Lessico del XXI Secolo", 03.04.2020

<https://gist.github.com/warmwaffles/8534618>, *PausableThreadPoolExecutor.java*, 03.02.2020.

https://en.wikipedia.org/wiki/Main_Page, Sorgente glossario, 03.04.2020

<https://vaadin.com/docs/v14/flow/binding-data/tutorial-flow-data-provider.html#lazy-loading-data-from-a-backend-service>, Soluzione Lazy Loading di percorsi, 23.03.2020

<https://vaadin.com/directory/component/filesystem-dataprovider-add-on>, *Filesystem DataProvider*, 05.03.2020

<https://farenda.com/java/java-regex-matching-ip-address/>, Soluzione regex per IP, 19.02.2020

<https://stackoverflow.com/questions/3263892/format-file-size-as-mb-gb-etc>, Rappresentazione grandezza dei duplicati, 03.04.2020

10 Glossario

Termine	Spiegazione
Repository o repo	Generico ambiente di storage, raggiungibile anche con un percorso web, dove vengono archiviati i pacchetti software che possono essere installati e aggiornati su un computer anche mediante operazioni programmate.
Versione LTS	Versione Long Term Support, è una politica di gestione del ciclo di vita di un prodotto in cui viene mantenuta una versione stabile del software per un periodo di tempo più lungo rispetto all'edizione standard.
Artefatti	In informatica, e in particolare in ingegneria del software, un artefatto è un sottoprodotto che viene realizzato durante lo sviluppo software. Sono

	artefatti i casi d'uso, i diagrammi delle classi, i modelli UML, il codice sorgente e la documentazione varia, che aiutano a descrivere la funzione, l'architettura e la progettazione del software.
REST API	L'architettura REST si basa su HTTP. Il funzionamento prevede una struttura degli URL ben definita che identifica univocamente una risorsa o un insieme di risorse e l'utilizzo dei verbi HTTP specifici per il recupero di informazioni (GET), per la modifica (POST, PUT, PATCH, DELETE) e per altri scopi (OPTIONS, ecc.)
Interfaccia di loopback	L'interfaccia di loopback viene utilizzata nelle reti TCP/IP per identificare la macchina locale su cui i programmi sono in esecuzione, detta anche localhost.
Nome canonico	Il nome della classe scritto nel seguente modo: package.className Questa convenzione segue le specifiche del linguaggio di Java
Filesystem	Un file system, indica informalmente un meccanismo con il quale i file sono posizionati e organizzati su dispositivi informatici utilizzati per l'archiviazione dei dati ad esempio unità di memoria di massa
Thread	Un thread o thread di esecuzione, in informatica, è una suddivisione di un processo in due o più filoni (istanze) o sottoprocessi che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocesso (multithreading) o multiprocesso o multicore.
View	Pagina o interfaccia
Branch (rame)	La diramazione, nel controllo della versione e nella gestione della configurazione del software, è la duplicazione di un oggetto sotto il controllo della versione (come un file di codice sorgente o un albero di directory) in modo che le modifiche possano avvenire in parallelo lungo più rami. La ramificazione generalmente implica anche la possibilità di unire o integrare successivamente le modifiche nel ramo padre.

11 Allegati

Elenco degli allegati:

- Diari di lavoro
- Codici sorgente
- Istruzioni di installazione del prodotto
- Manuale di utilizzo
- Il Quaderno dei compiti
- Prodotto
- Il Gantt Preventivo e Consuntivo
- I screenshot delle interfacce

