


Deduplicatore di files

1	Introduzione.....	3
1.1	Informazioni sul progetto.....	3
1.2	Abstract.....	3
1.3	Scopo.....	3
2	Analisi.....	4
2.1	Analisi del dominio.....	4
2.2	Analisi e specifica dei requisiti.....	4
2.3	Use case.....	9
2.4	Pianificazione.....	10
2.5	Analisi dei mezzi.....	11
2.5.1	Software.....	11
2.5.2	Hardware.....	11
3	Progettazione.....	11
3.1	Design dell'architettura del sistema.....	11
3.1.1	Deduplicator.....	12
3.1.2	DeduplicatorGUI.....	19
3.2	Design dei dati e database.....	22
3.3	Design delle interfacce.....	23
3.4	Design procedurale.....	30
4	Implementazione.....	31
4.1	Deduplicator.....	31
4.1.1	SecurityConfig.java.....	31
4.1.2	Controller.....	32
4.1.3	Entity.....	40
4.1.4	Repository.....	40
4.1.5	Scanner.....	42
4.1.6	Validator.....	46
4.1.7	Resources.....	48
4.2	DeduplicatorGUI.....	49
4.2.1	Communication.....	49
4.2.2	MainJFrame.....	54
5	Test.....	56
5.1	Protocollo di test.....	56
5.2	Risultati test.....	60
5.3	Mancanze/limitazioni conosciute.....	61
6	Consuntivo.....	62
7	Conclusioni.....	63
7.1	Sviluppi futuri.....	63
7.2	Considerazioni personali.....	63
8	Sitografia.....	63
9	Allegati.....	64

	SAMT - Sezione Informatica	Pagina 3 di 64
	Deduplicatore di files	

1 Introduzione

1.1 Informazioni sul progetto

Si tratta di creare un applicazione che serve a gestire i duplicati di file in uno o più percorsi definiti dall'utente. L'intenzione è di creare una applicazione che lavora in background e un'interfaccia grafica per permettere la configurazione all'utente. Il docente responsabile è Geo Petrini, il termine del progetto è il 20 dicembre.

1.2 Abstract

As the quantity and diversity of files created by an average user increases it becomes hard to keep track of all of the files a user has on their PC. This program has the task to simplify the search of duplicate files in order to save the users available space on disk. The project can be separated in 2 parts: the actual deduplicator that will run in the background as a service and a GUI made for the user to configure and manage the service. The program might also help the user realize how many files he has. The GUI part of the program can also be used by a SysAdmin to manage a small local network of PC's that have the service running. The scanning operation can be executed on demand or it can be set on a schedule.

1.3 Scopo

Lo scopo del progetto è di creare un programma per gestire i file duplicati e che ha la possibilità di essere eseguito su multiple piattaforme (Windows/Linux/MacOS). Il progetto sarà diviso in 2 parti: La parte del servizio che lavora in background e la parte della GUI dove verrà eseguita la configurazione del servizio da parte dell'utente.

2 Analisi

2.1 Analisi del dominio

Attualmente sul mercato esistono dei tool per la deduplicazione, i più conosciuti sono: *CloneSpy*, *Duplicate Cleaner Pro/Free*, *Dupscout*, *Advanced Duplicates Finder*, *Duplicate Finder*, *Auslogistics Duplicate File Finder*, *Fast Duplicate File Finder*, *Anti-Duplicate* e altri, ma la maggior parte di loro è a pagamento oppure non si adegua ai requisiti imposti, cioè lavorare in background e essere configurabili da remoto. Il programma deve inoltre poter essere utilizzato sia per scopo personale che per scopo di piccole aziende dove un sysadmin gestisce una piccola rete. Per usare il prodotto l'utente avrà bisogno di conoscenze minime su come utilizzare un pc.

2.2 Analisi e specifica dei requisiti

I requisiti per questo progetto sono stati definiti dal docente responsabile Geo Petrini. Il programma sarà suddiviso in 2 parti: la parte del servizio che accetterà i comandi e che infine eseguirà le scansioni e la parte della GUI che sarà utile all'utente per configurare e usare la parte del servizio. Il servizio deve eseguire la scansione in modo Multithreaded basandosi su un file di configurazione per sapere che percorsi scansionare. Il servizio alla fine della scansione deve produrre un rapporto tramite quale l'utente potrà visualizzare i file duplicati e decidere cosa fare. Non è prevista una gestione degli accessi / utenti al servizio, anche se in futuro potrebbe essere aggiunta. La comunicazione tra il servizio e la GUI deve avvenire in modo sicuro utilizzando HTTPS con autenticazione.

ID: REQ-01	
Nome	Comunicazione sicura
Priorità	1
Versione	1.0
Note	La comunicazione tra il servizio e la GUI deve avvenire in modo sicuro tramite richieste HTTPS + autenticazione
Sotto requisiti	
001	Si necessita un webserver HTTPS dal lato del servizio
002	Si necessita una coppia di chiave pubblica e privata per l'autenticazione

ID: REQ-02	
Nome	Comunicazione usando il protocollo REST
Priorità	1
Versione	1.0
Note	La comunicazione tra il servizio e la GUI deve avvenire usando il protocollo REST
Sotto requisiti	
001	Si necessita di un approccio MVC

ID: REQ-03	
Nome	Creazione del rapporto
Priorità	1
Versione	1.0
Note	Alla fine della scansione il programma deve generare un rapporto


ID: REQ-04	
Nome	L'esecuzione delle azioni
Priorità	1
Versione	1.0
Note	Questo requisito dipende dal requisito REQ-04, perché solo dopo che un rapporto sia stato generato si possono impostare le azioni da eseguire per i file duplicati trovati. Si avrebbe anche bisogno dei permessi di root, a dipendenza dei percorsi, per poter eseguire alcune operazioni
Sotto requisiti	
001	La possibilità di eseguire le azioni programmaticamente
002	La messa in coda delle azioni da eseguire

ID: REQ-05	
Nome	GUI per utente per il controllo
Priorità	1
Versione	1.0
Note	Creare una GUI per l'utente
Sotto requisiti	
001	Possibilità di inserire i percorsi
002	Possibilità di inserire dei percorsi da escludere (whitelist)
003	Possibilità di gestire le scansioni in corso (vedi REQ-07)

ID: REQ-06	
Nome	Messa in pausa della scansione
Priorità	2
Versione	1.0
Note	La possibilità di fermare o mettere in pausa una scansione.

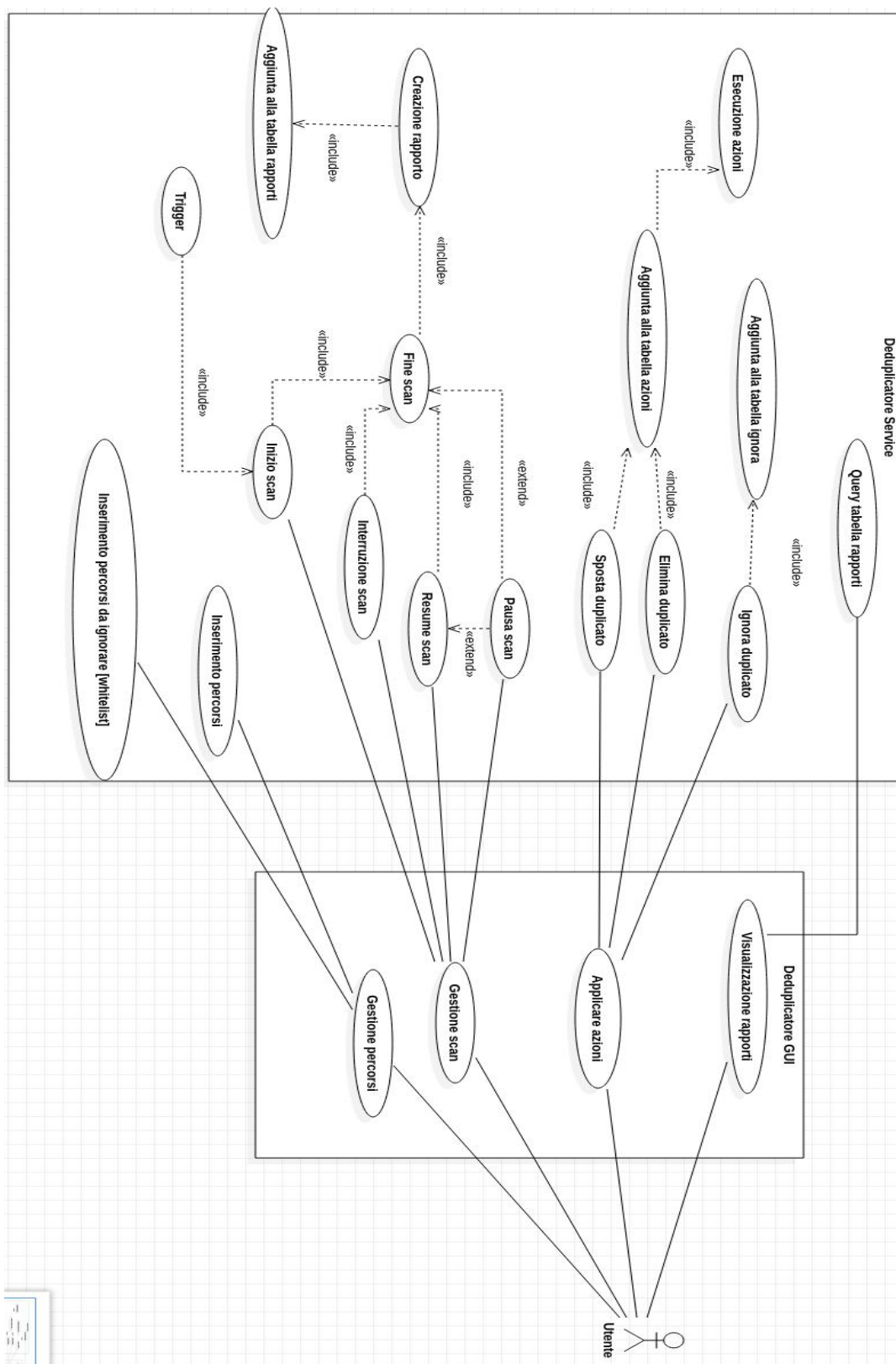
ID: REQ-07	
Nome	Gestione dei rapporti
Priorità	2
Versione	1.0
Note	La possibilità di salvare, ricaricare un rapporto passato e la continuazione di scansione di un rapporto non finito.

ID: REQ-08	
Nome	Un scheduler delle scansioni
Priorità	2
Versione	1.0
Note	La possibilità di pianificare l'esecuzione delle scansioni.

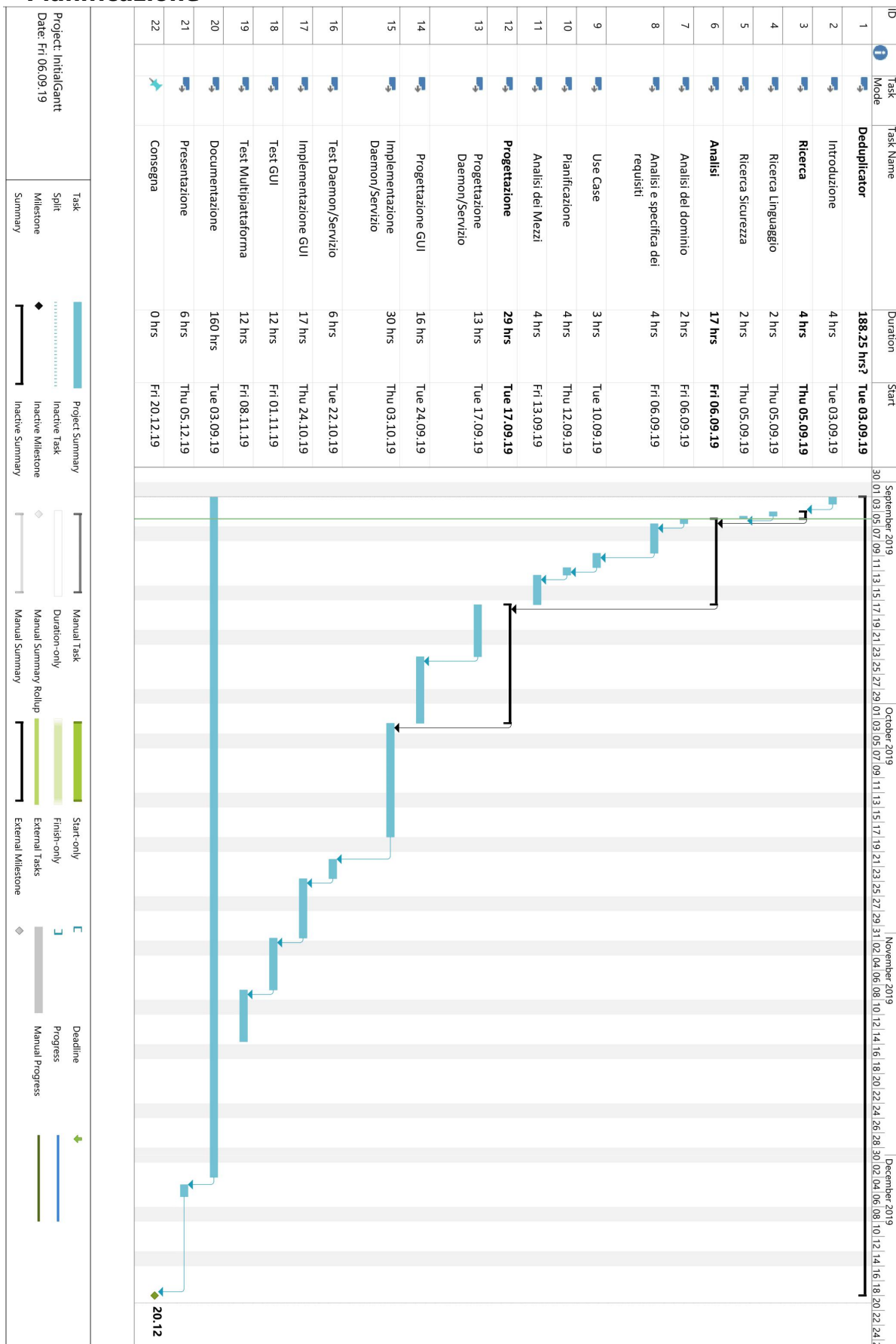
	SAMT - Sezione Informatica	Pagina 7 di 64
	Deduplicatore di files	


ID: REQ-09	
Nome	Rilevazione in tempo reale
Priorità	3
Versione	1.0
Note	Usare i trigger di sistema per trovare nuovi duplicati in tempo reale.

2.3 Use case



2.4 Pianificazione



	<div style="text-align: center;"> SAMT - Sezione Informatica </div> <hr/> <div style="text-align: center;"> Deduplicatore di files </div>	Pagina 11 di 64
--	---	-----------------

2.5 Analisi dei mezzi

Per la creazione di questo progetto la scuola mi mette a disposizione tutti i tool che sono disponibili a scuola e 1 accesso presso l'hosting interno per caricare il progetto.

2.5.1 Software

Per lo sviluppo userò il framework Spring Boot 2.1.8, MySQL **(5.7.28-0ubuntu0.19.04.2)**, VSCode 1.41.1.

Per i test Postman 7.8.0.

Il framework che utilizzerò dipende da Java 11 e Gradle 4.4.1 per la compilazione del progetto

Visual paradigm 16.1 per la generazione degli UML.

2.5.2 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

GPU: Intel UHD Graphics 620

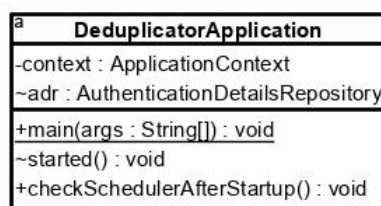
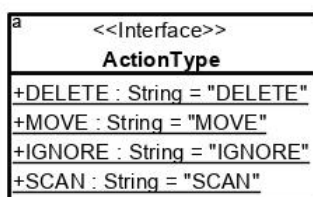
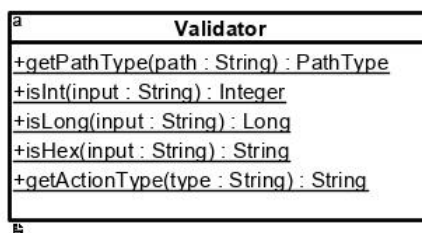
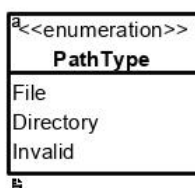
OS: Ubuntu 18.04.3 LTS / Gnome 3.28.2

3 Progettazione

3.1 Design dell'architettura del sistema

Di seguito verranno rapportati tutti diagrammi UML delle classi suddivisi per progetto e poi per package.

3.1.1 Deduplicator



3.1.1.1 Package controller

a	FileController
~	FileRepository : FileRepos...
~	reportRepository : Report...
+getAll()	: Iterable<File>
+getId()	: String : Object

a	SchedulerController
-context :	ApplicationContext
-schedulerRepository :	SchedulerRepository
+getAll()	: Iterable<Scheduler>
+getId()	: String : Object
+insert(monthly : String, weekly : String, repeated : String, timeStart : ...	
+delete(id : String) : Object	
+getPosition(number : Integer, max : int) : Integer	
+getPositions(number : Integer, max : int) : List<Integer>	

a	LoginController
-USERNAME_LENGTH :	int = 4
-PASSWORD_LENGTH :	int = 8
-adr :	AuthenticationDetailsRepository
+checkLogin()	: Message
+insert(username : String, password : String) : Object	
+delete(username : String) : Object	

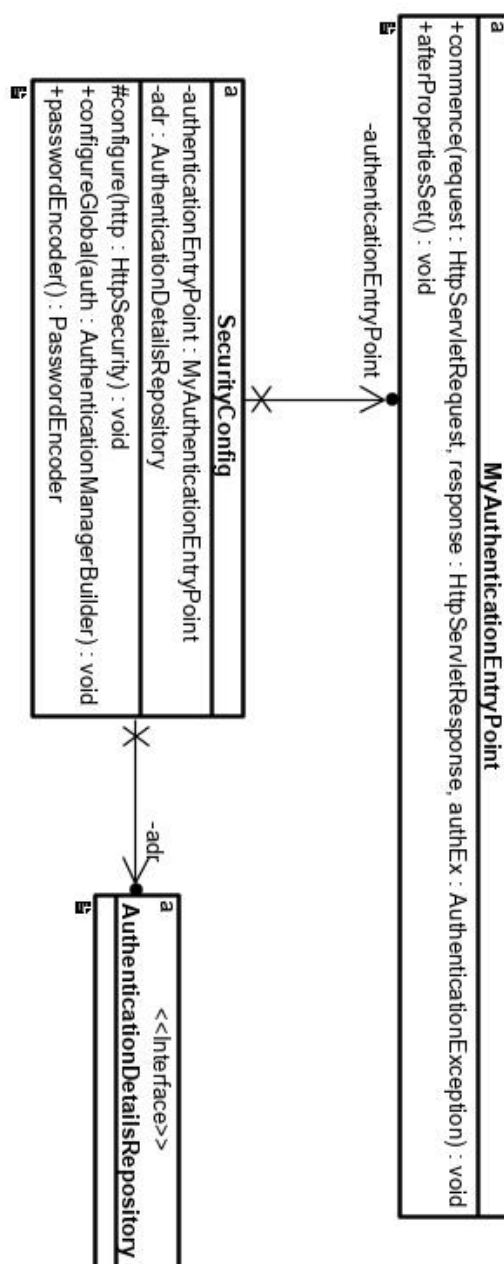
a	ActionController
-context :	ApplicationContext
-actionRepository :	ActionRepository
-schedulerRepository :	SchedulerRepository
-adr :	AuthenticationDetailsRepository
+getAll()	: Iterable<Action>
+getId()	: String : Object
+executeActions()	: Object
+insert(type : String, path : String, newPath : ...	
+deleteAction(id : String) : Object	
+checkPath(path : String) : Message	

a	ScanController
-context :	ApplicationContext
-reportRepository :	ReportRe...
-adr :	AuthenticationDetailsR...
-gpr :	GlobalPathRepository
-currentScan :	ScanManager
-report :	Report = null
+start(threadCount : Integer) : ...	
+stop() :	Object
+pause()	: Message
+resume()	: Message
+getStatus()	: Object
+scanFinished()	: void
-destroyScanManager()	: void

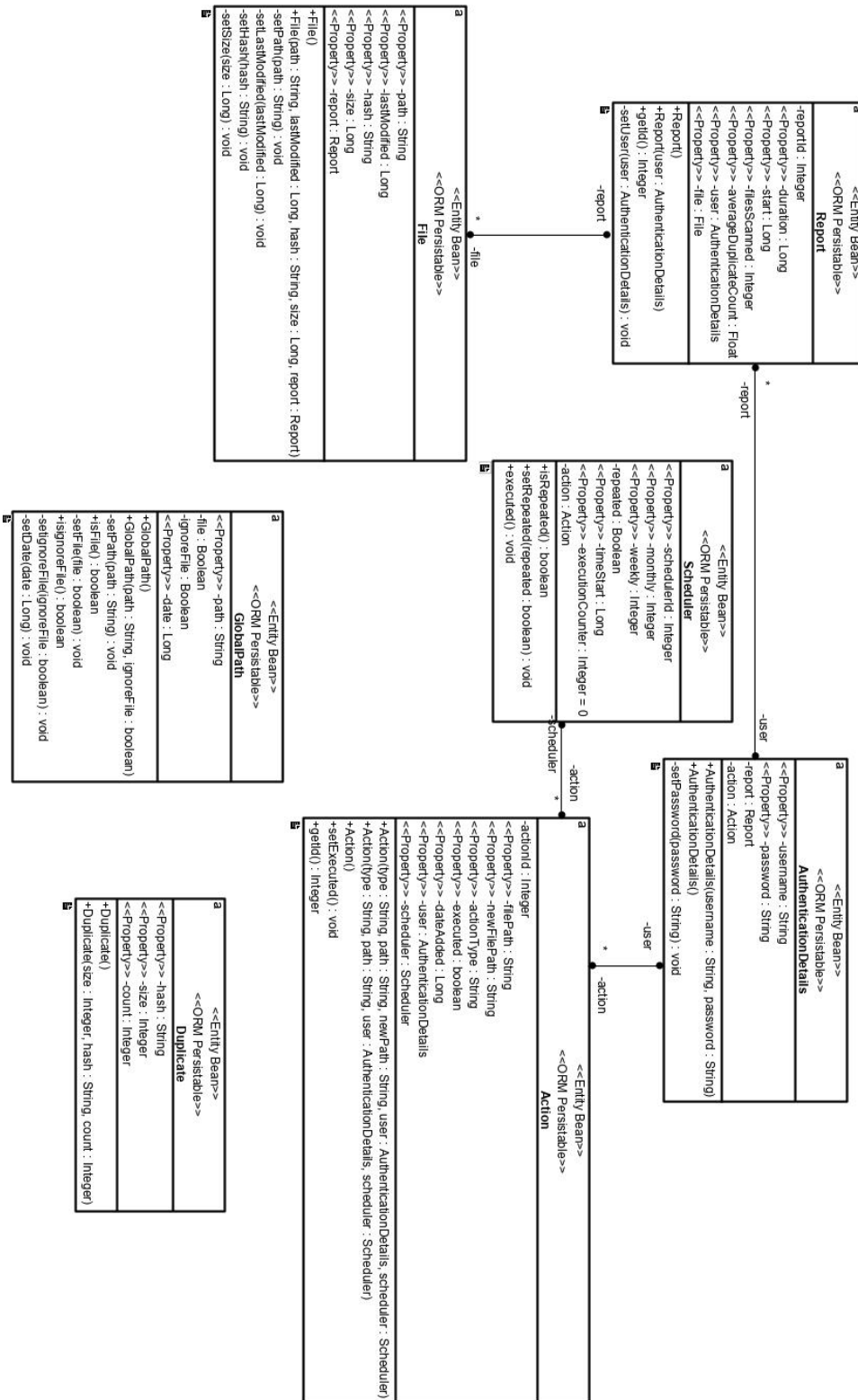
a	ReportController
~reportRepository :	ReportRepository
~duplicateRepository :	DuplicateRepository
~fileRepository :	FileRepository
+getAllReports()	: Iterable<Report>
+getAllReportsReduced()	: Object
+getLatestReport()	: Report
+getReportById(id : String) : Object	
+getDuplicateByReportId(id : String) : Object	
+getFileByHash(id : String, hash : String) : O...	

a	PathController
~gpr :	GlobalPathRepository
+getAll()	: Iterable<GlobalPath>
+get(path : String) : Object	
+insert(path : String, ignorePath : String) : Object	
+delete(path : String) : Object	

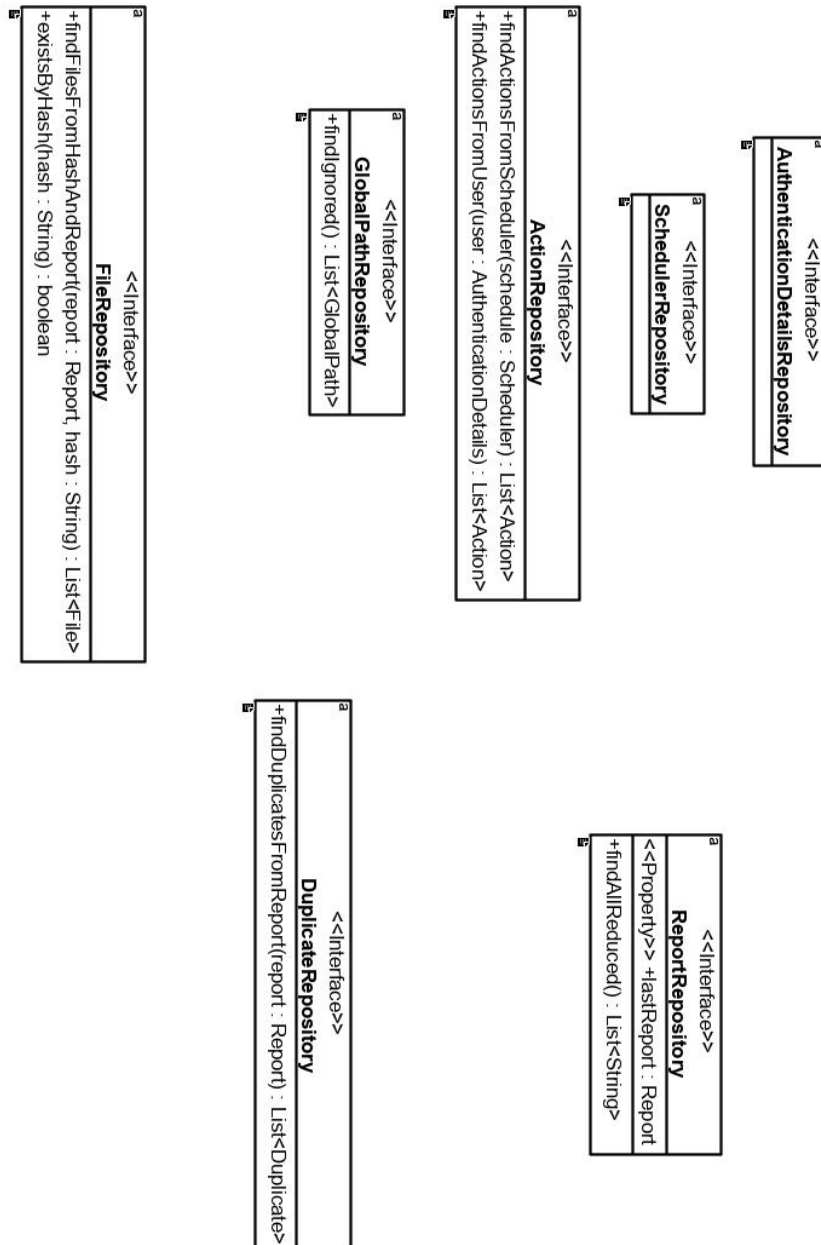
3.1.1.2 Package config



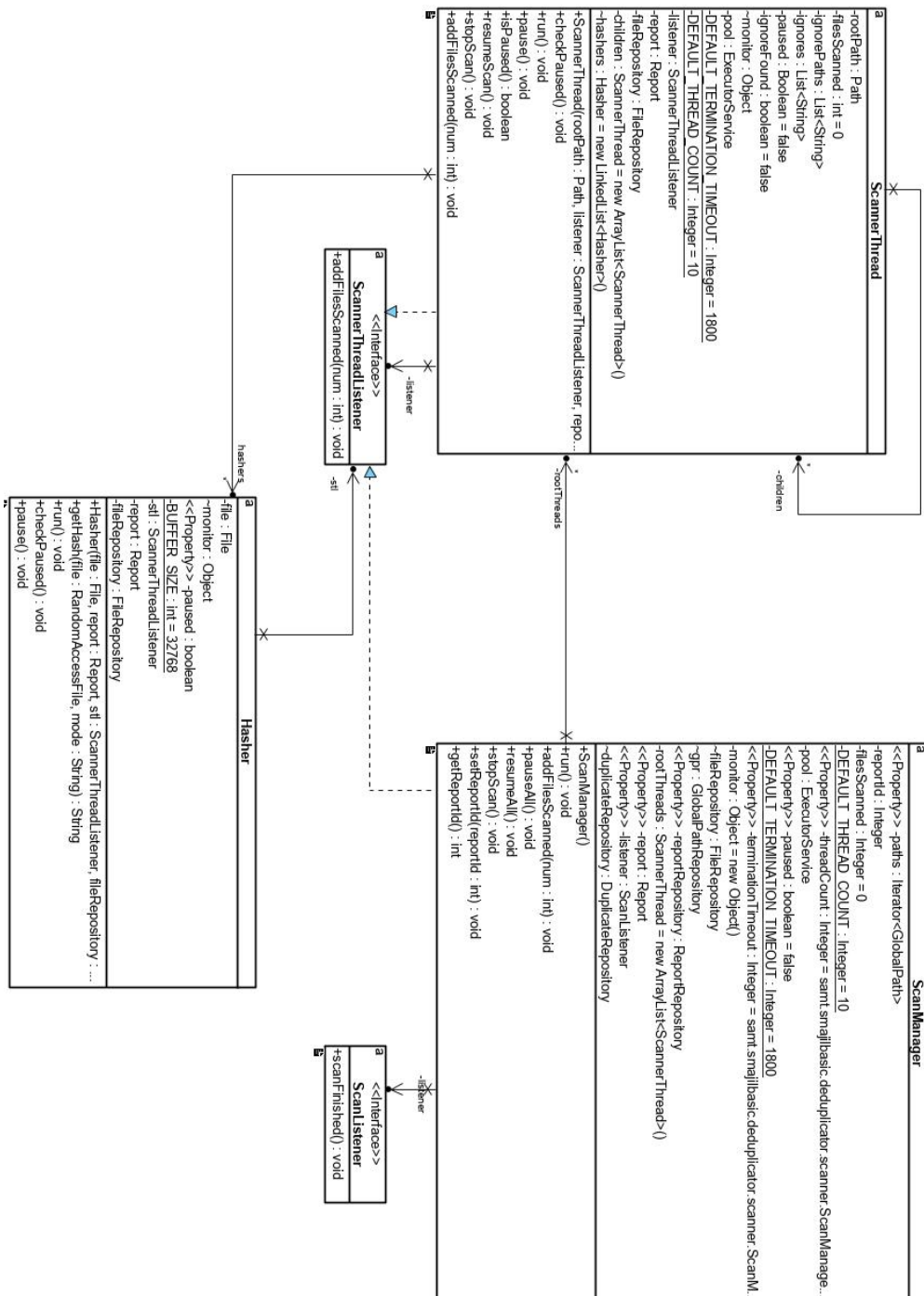
3.1.1.3 Package entity



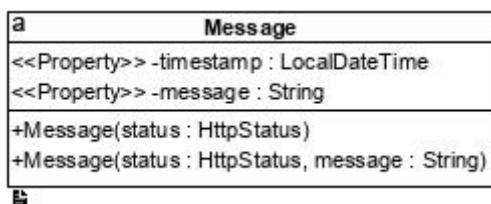
3.1.1.4 Package repository



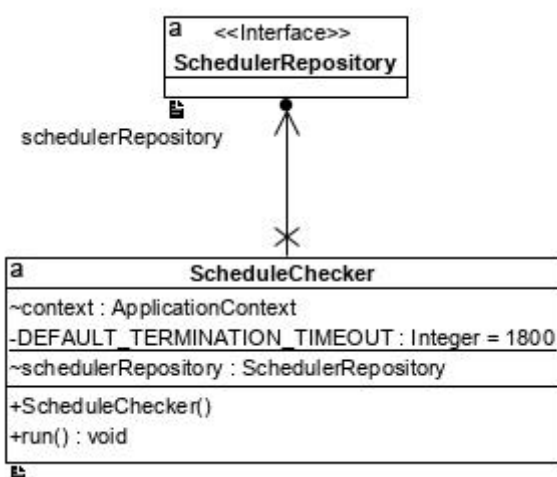
3.1.1.5 Package scanner



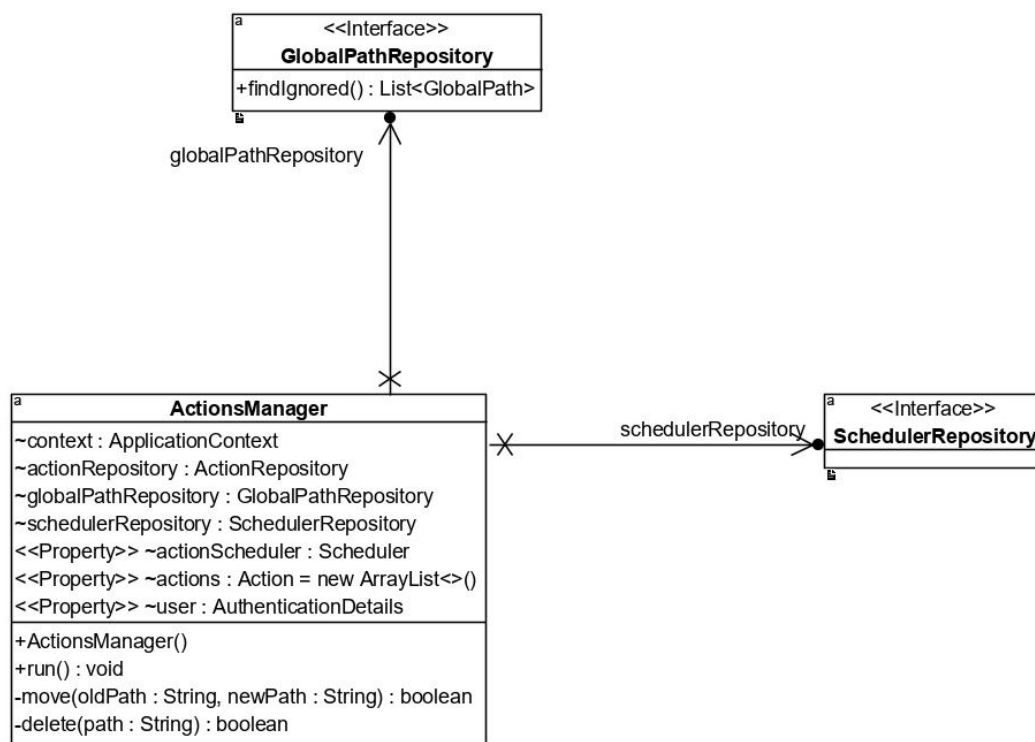
3.1.1.6 Package exception



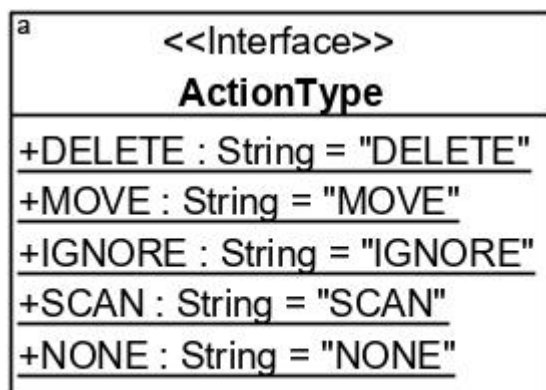
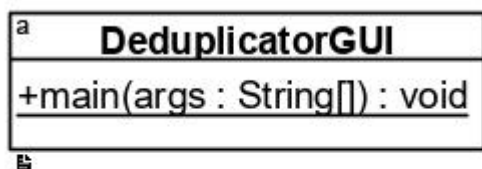
3.1.1.7 Package timer



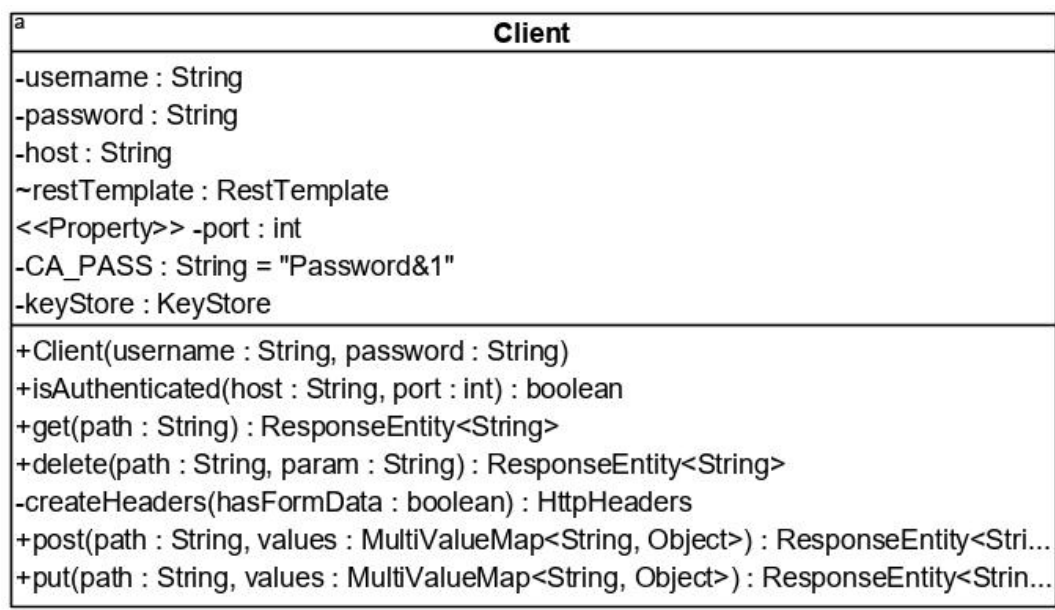
3.1.1.8 Package worker



3.1.2 DeduplicatorGUI



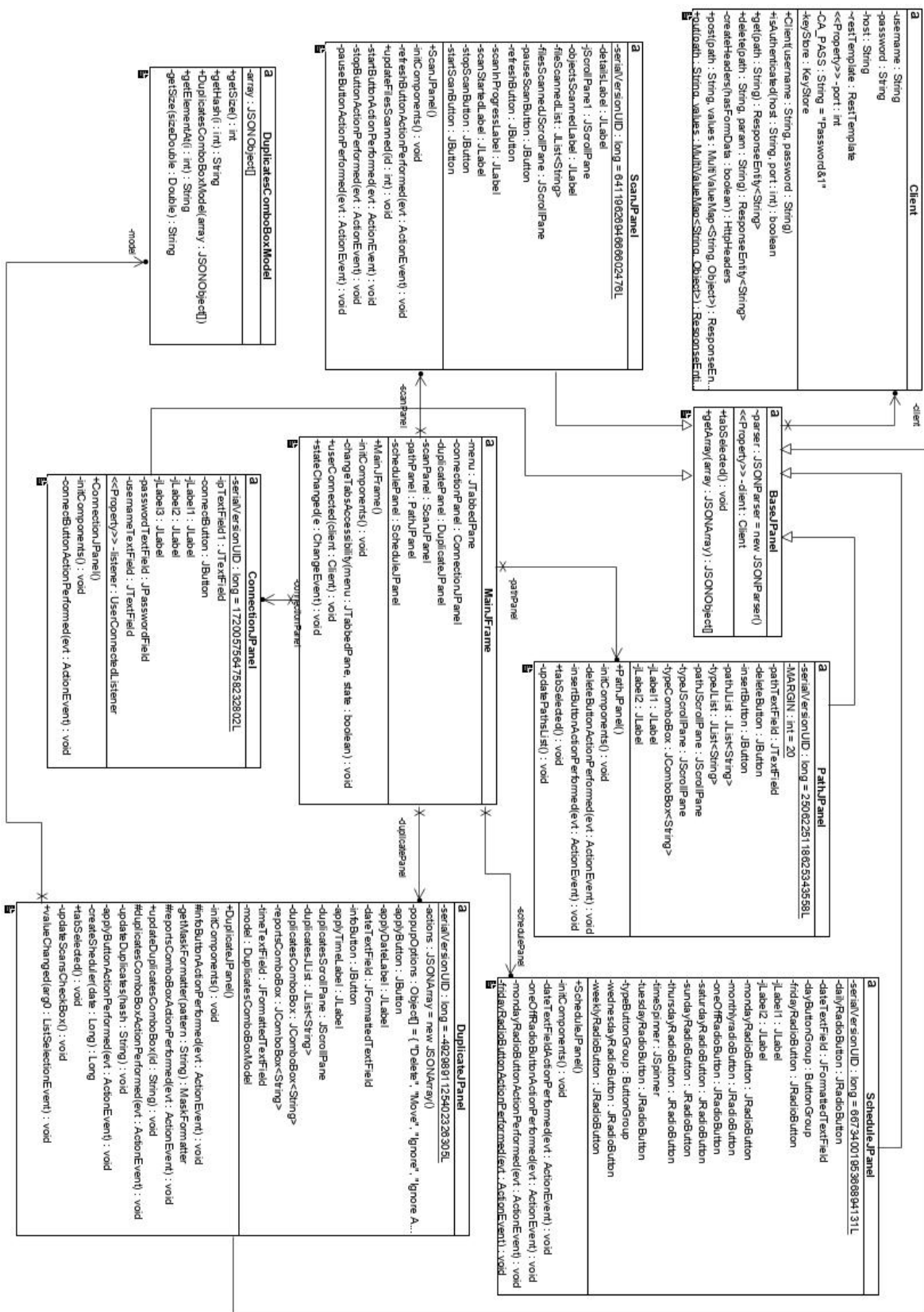
3.1.2.1 Package communication



3.1.2.2 Package listeners



3.1.2.3 Package layout



La immagine del package layout verrà messa tra gli allegati in formato più grande

3.2 Design dei dati e database

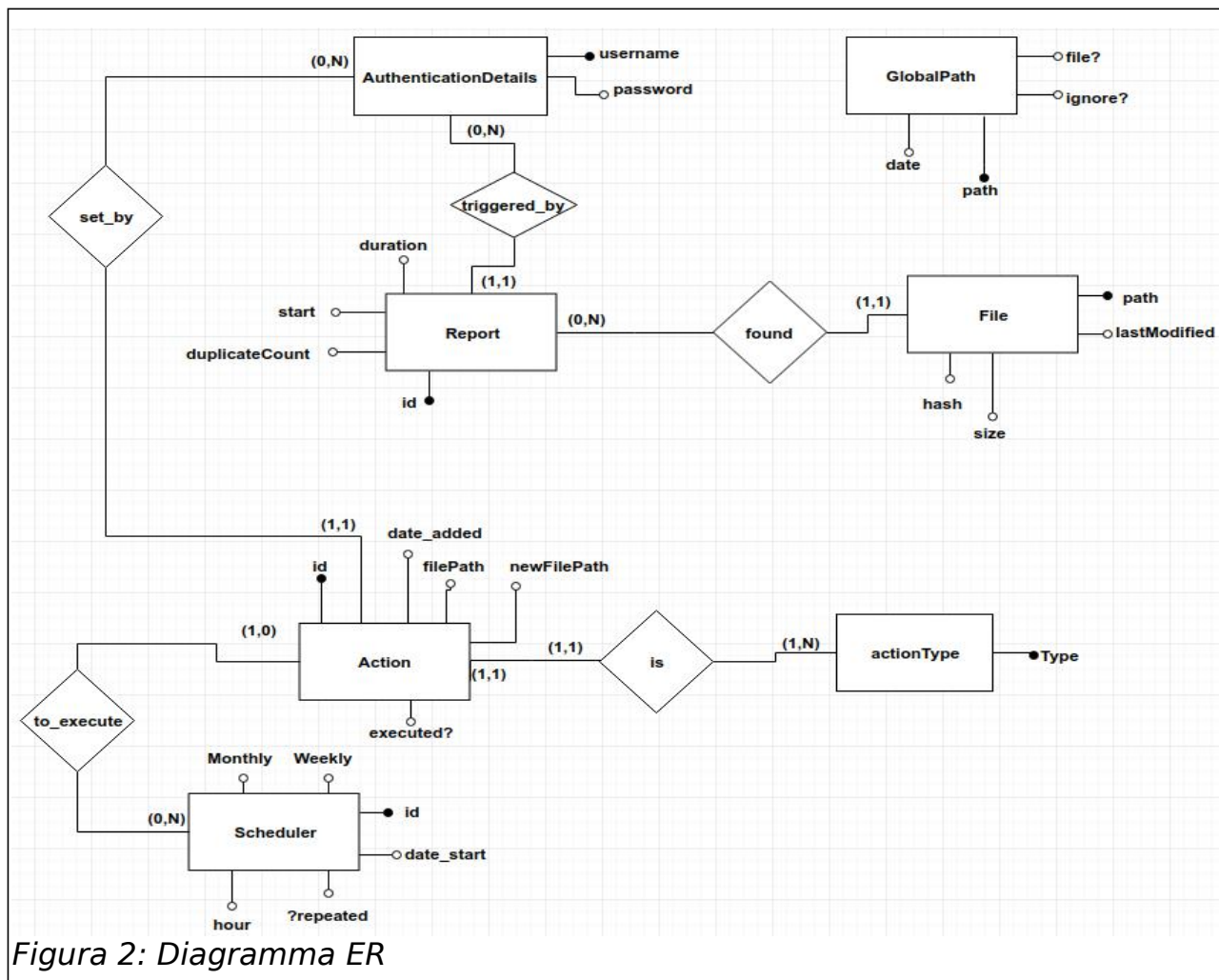


Figura 2: Diagramma ER

Il diagramma ER è fatto da 7 entità e 4 relazioni.

L'entità GlobalPath conterrà i percorsi da scansione o da ignorare che il servizio userà durante la scansione.

L'entità Report contiene informazioni sui report creati alla fine della scansione, contiene il numero di duplicati, da chi è stato eseguito (utente o scheduler), il timestamp di quando è stato eseguito e un id.

L'entità File verrà usata per salvare informazioni sui file scansionati, contiene il percorso con nome del file, la data dell'ultima modifica, il hash in MD5 (32 Byte) del file e la grandezza del file. Inoltre contiene l'informazione in quale report è stato scoperto il file.

L'entità ActionType contiene il tipo di azione che verrà applicata ai file (ignora, elimina, sposta)

L'entità Action contiene le azioni da eseguire dopo che l'utente abbia revisionato un report, ogni azione ha un id, una data d'aggiunta, il percorso del file, se l'azione è stata eseguita e dopo essere eseguita viene aggiunto un riferimento segnalando quale schedule nella tabella Scheduler abbia fatto eseguire l'azione e l'utente che ha scelto quella azione.

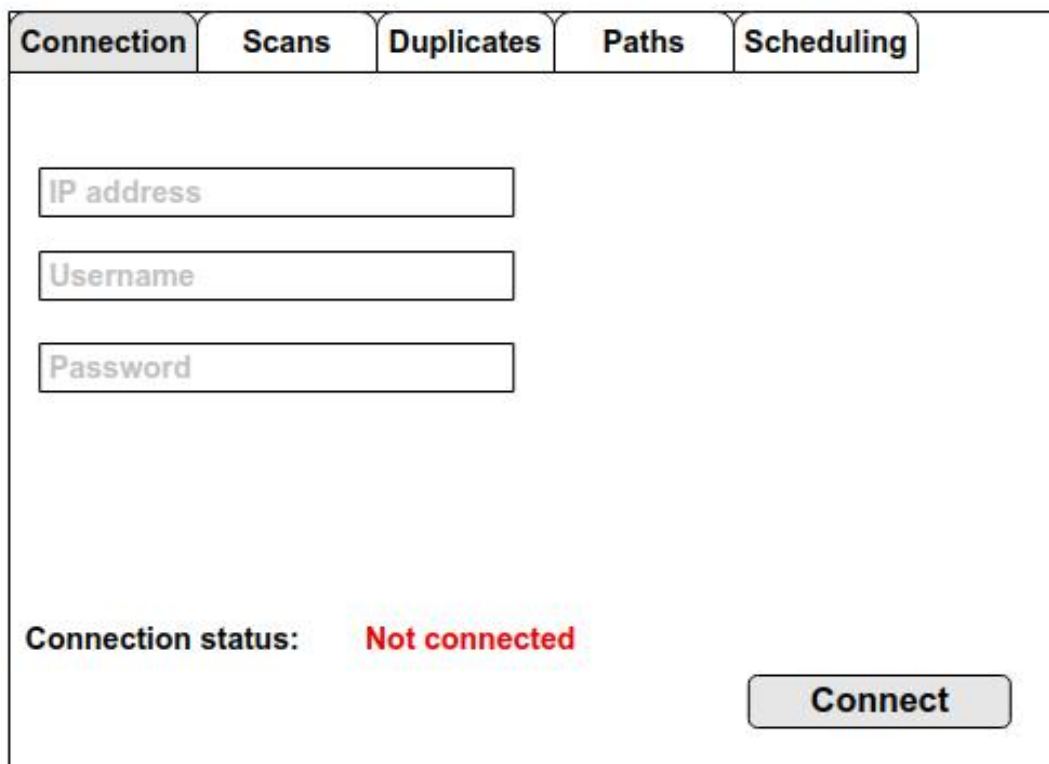
L'entità Scheduler contiene informazioni sulle operazioni programmate che dovranno essere eseguite, verrà utilizzato da una parte del servizio per controllare quali schedule sono da eseguire e quali no.

La tabella AuthenticationDetails contiene il username e la password per poter verificare le credenziali al momento della creazione della connessione.

Report contiene informazioni sulla scansione, quando è stata avviata, quanto è durata, i duplicati trovati e da chi è stata avviata, nel caso che essa venga avviata tramite lo scheduler, l'utente che l'ha avviata verrà impostato come quello di default.


3.3 Design delle interfacce

In seguito saranno rappresentati i mockup delle interfacce della GUI, il servizio non ha alcuna interfaccia utente.



Connection	Scans	Duplicates	Paths	Scheduling
<div style="margin-bottom: 10px;"> <input style="width: 100%;" type="text" value="IP address"/> </div> <div style="margin-bottom: 10px;"> <input style="width: 100%;" type="text" value="Username"/> </div> <div style="margin-bottom: 10px;"> <input style="width: 100%;" type="password" value="Password"/> </div> <div style="display: flex; justify-content: space-between; align-items: center;"> <div> Connection status: Not connected </div> <div> <input style="border: 1px solid gray; padding: 5px 15px;" type="button" value="Connect"/> </div> </div>				

Figura 3: L'interfaccia della Connessione

	SAMT - Sezione Informatica	Pagina 24 di 64
	Deduplicatore di files	

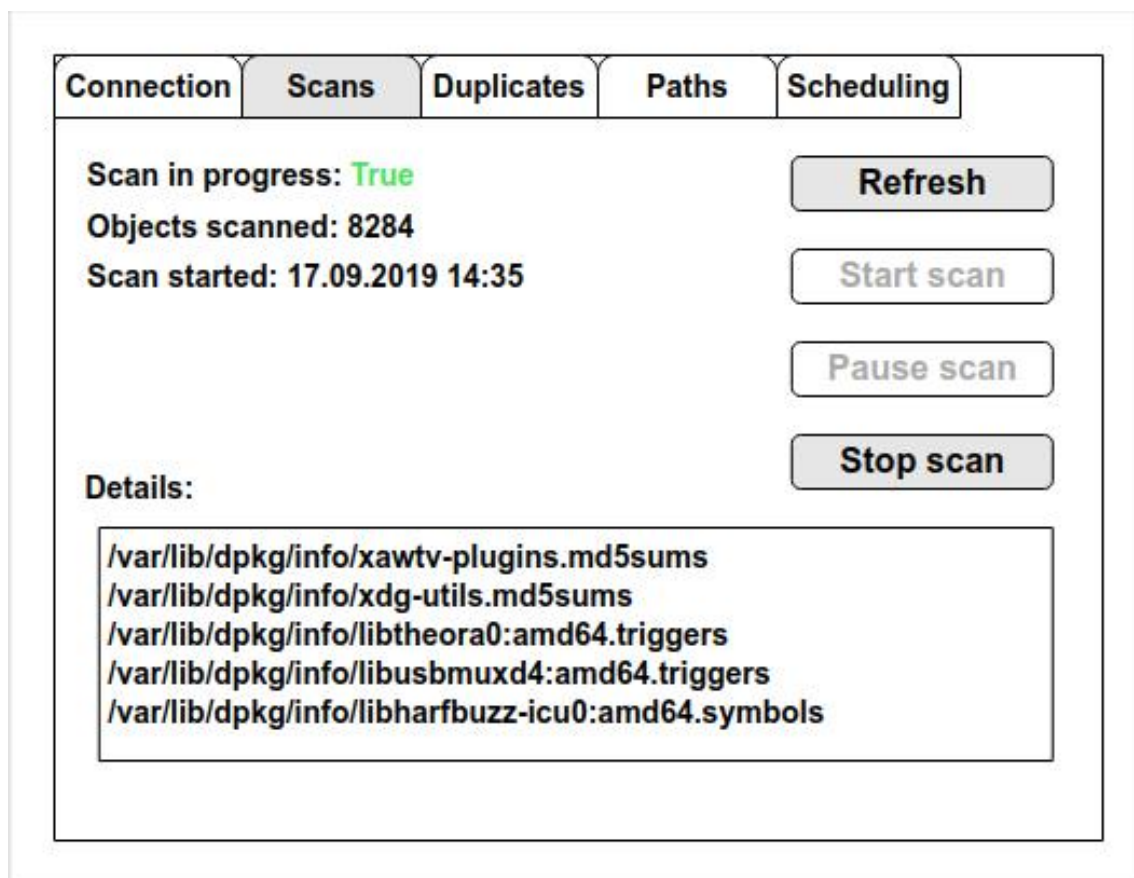


Figura 4: L'interfaccia che contiene le informazioni sulle scansioni

Connection

Scans

Duplicates

Paths

Scheduling

Scan 15.09.2019 11:30

info

Duplicates found: 12

Total files: 29

Duplicate 1:

Hash: 7ead3c169f4b3b2dac7ab45e6c32f6e3

Size: 19KB

Ignore all

▼ Path	▼ Name	▼ Last modified	▼ Action
/home/john/Documents/	filename.txt	11-09-2019	Delete ▼
/home/john/	nicknames.txt	04-05-2019	Ignore ▼
/home/john/Pictures/emails/attachments/	filename.txt	26-08-2019	Delete ▼
/home/john/randomFiles/	todo.txt	31-08-2019	Move ▼

Duplicate 2:

Hash: 20905d243ff3538d2082b4443ede9998

Size: 12KB

Ignore all

▼ Path	▼ Name	▼ Last modified	▼ Action
/home/john/Documents/	another.txt	08-07-2019	Delete ▼
/home/john/	dummy1.txt	05-04-2019	Move ▼

Apply date:

12/10/2019

Apply apply hour:

2:00

Apply

Figura 5: Interfaccia contenente i rapporti delle scansioni

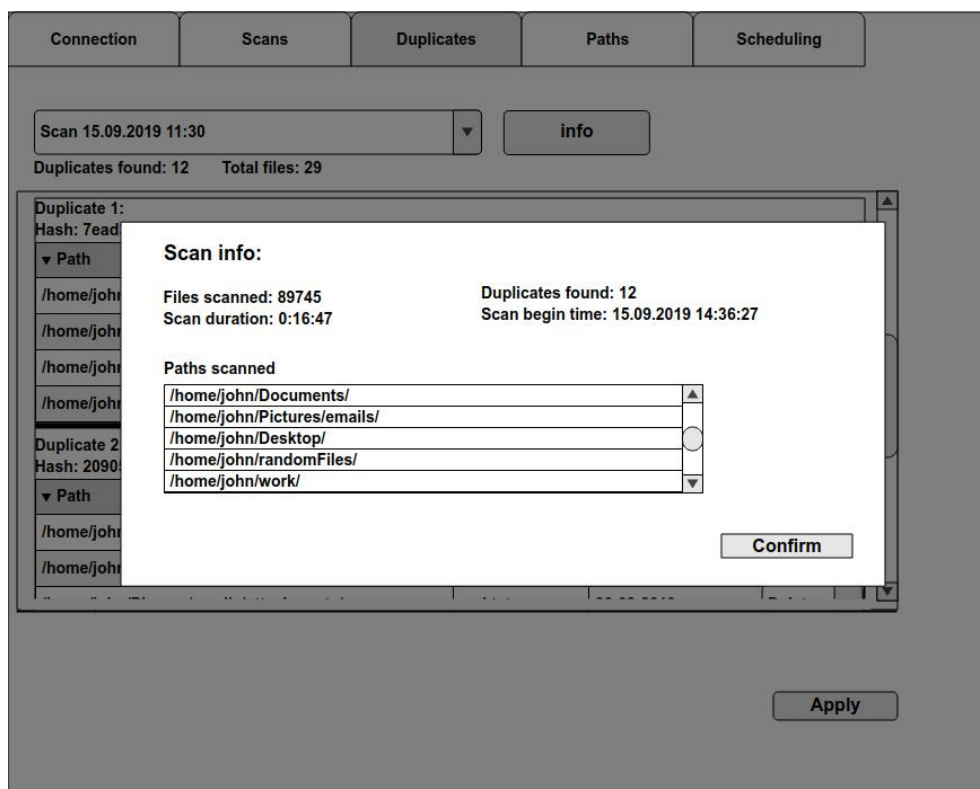


Figura 6: La visuale che si vede quando si clicca il bottone

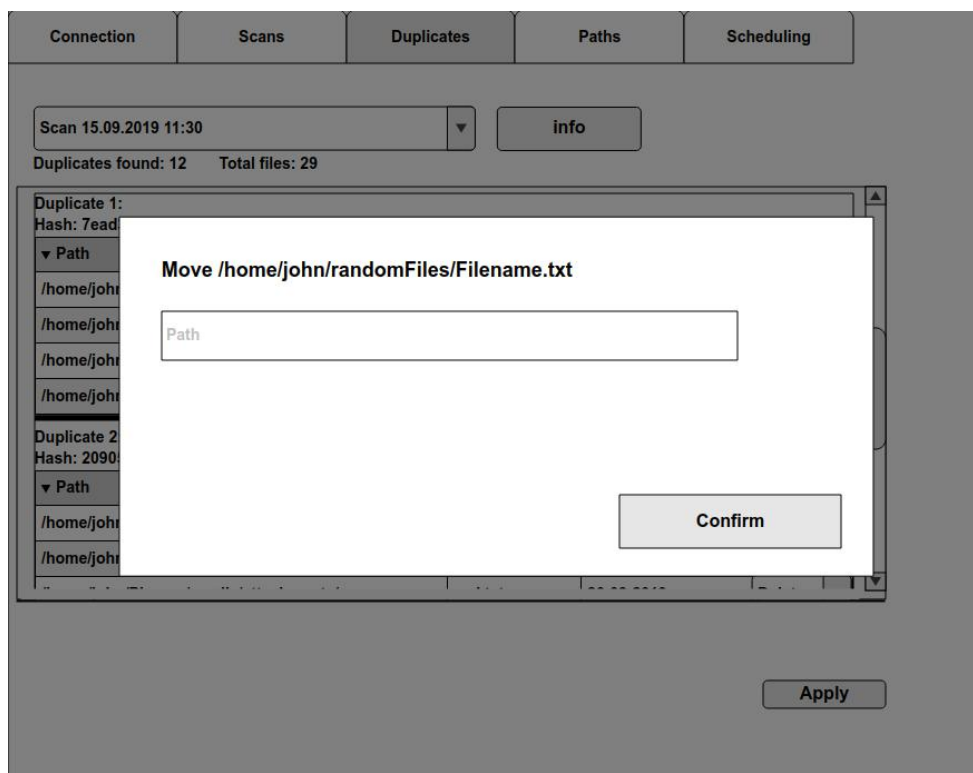


Figura 7: Interfaccia che si vede quando si sceglie di muovere un file

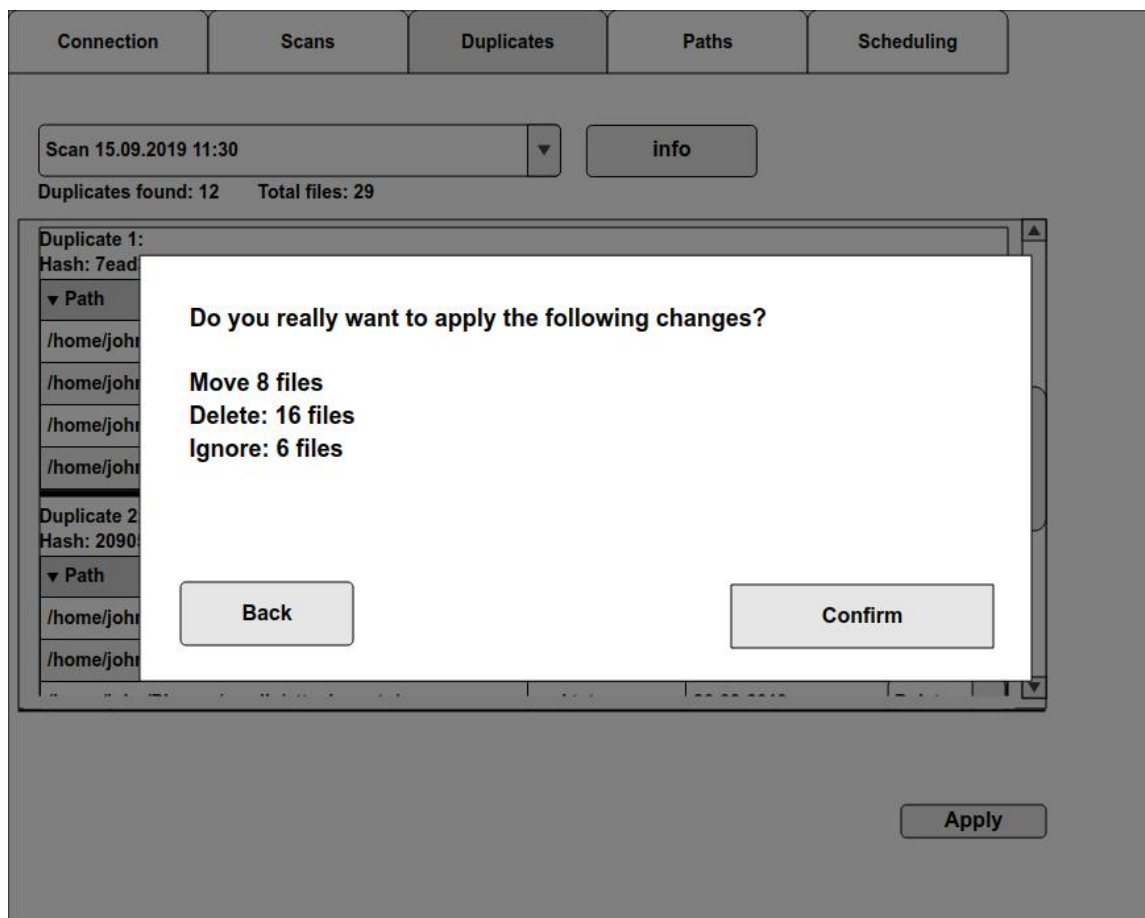


Figura 8: Il riassunto delle operazioni, questa interfaccia è visibile quando si schiaccia il tasto apply

The screenshot shows the 'Paths' tab of the SAMT software. At the top, there are five tabs: 'Connection', 'Scans', 'Duplicates', 'Paths' (selected), and 'Scheduling'. Below the tabs, there is a 'Path' input field, an 'Ignore' dropdown menu, and an 'Insert' button. Below this, a section titled 'Paths:' contains a table with the following data:

/home/John/Documents/Backup/Filename.pdf	Ignore
/home/John/Documents/Backup/OtherFile.odg	Ignore
/home/John/Documents/Backup/BigFile.odg	Ignore
/home/John/Documents/A_Folder/	Scan
/home/John/Documents/BigFolder	Ignore
/home/John/Documents/	Scan

Figura 9: L'interfaccia contenete i percorsi da ignorare o da scansionare

The screenshot shows the 'Scheduling' tab of the SAMT software. At the top, there are five tabs: 'Connection', 'Scans', 'Duplicates', 'Paths', and 'Scheduling' (selected). Below the tabs, a section titled 'Plan new scan:' contains the following options:

Scan start:
 Date: 4/22/2012 (with a calendar icon)
 Time: 12:00 (with up/down arrows)

Frequency:
☐ One off
☐ Daily
☒ Weekly
☐ Monthly

Days:
☐ Monday
☒ Tuesday
☐ Wednesday
☒ Thursday
☐ Friday
☒ Saturday
☒ Sunday

At the bottom right, there is a 'Confirm' button.

Figura 10: L'interfaccia contenente le informazioni sulle scansioni pianificate

3.4 Design procedurale

Nel seguente diagramma di flusso si può vedere come lavorerà la parte del servizio che sta dietro alla GUI

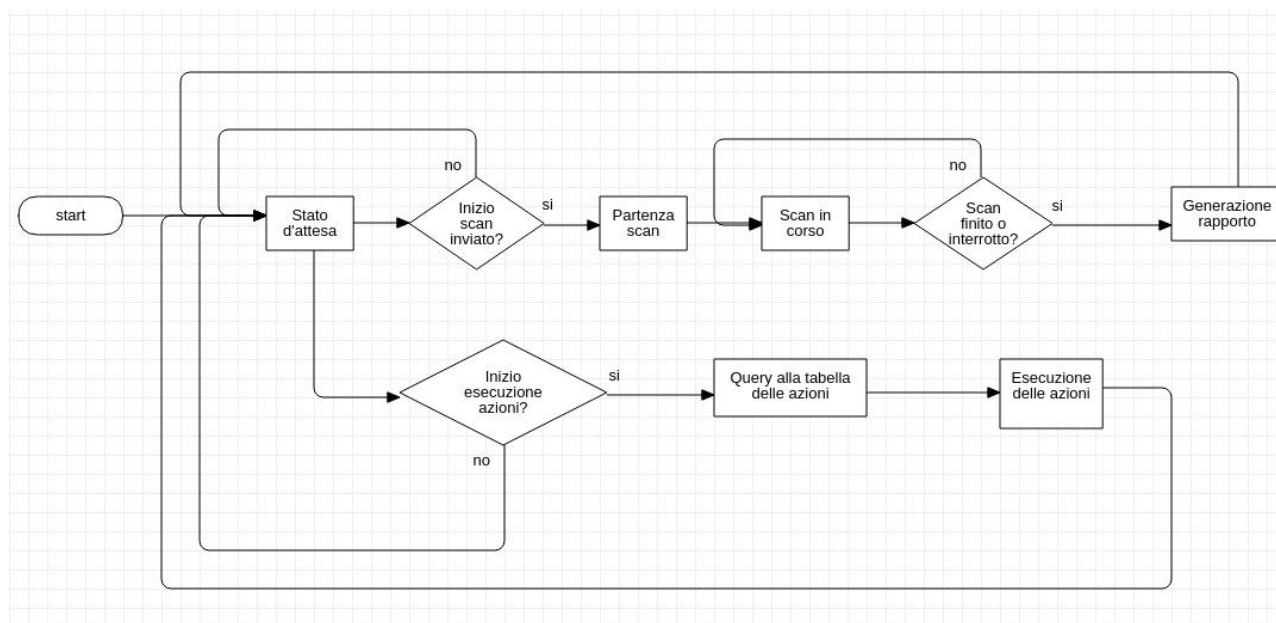


Figura 11: Diagramma di flusso servizio

4 Implementazione

Il progetto è stato diviso in due parti: deduplicator (il servizio) e deduplicatorGUI (l'interfaccia utente)

4.1 Deduplicator

Accesso di default tramite autenticazione BASIC:

Username: admin
Password: admin

Si possono inserire o eliminare utenti tramite il controller LoginController. Gli utenti non si differenziano a livello dei permessi.

4.1.1 SecurityConfig.java

La classe SecurityConfig configura il webserver spring ad obbligare i client a usare HTTPS e l'autenticazione BASIC tramite il metodo **configure**.

```
@Override
protected void configure(HttpSecurity http) throws Exception
{
    http
        .csrf().disable()
        .authorizeRequests().anyRequest().authenticated() // richiesta autorizzazione per ogni controller e ogni tipo di richiesta
        .and()
        .httpBasic() //abilita autenticazione basic
        .and()
        .exceptionHandling().authenticationEntryPoint(authenticationEntryPoint) // definizione punto d'entrata in caso che l'utente non è autenticato
        .and().requiresChannel().anyRequest().requiresSecure(); // richiesta utilizzo protocollo sicuro (TLS) per ogni tipo di
}
```

Inoltre la classe SecurityConfig implementa un attributo di tipo MyAuthenticationEntryPoint, questo attributo contiene un metodo **commence** che risponde ad una richiesta senza autenticazione con un messaggio d'errore personalizzato **Message**.

```
@Override
public void commence(
    HttpServletRequest request, HttpServletResponse response, AuthenticationException authEx)
    throws IOException, ServletException {
    response.addHeader("WWW-Authenticate", "Basic realm=" + getRealmName() + "");
    response.addHeader("Content-Type", "application/json");
    response.setStatus(HttpStatus.SC_UNAUTHORIZED);
    PrintWriter writer = response.getWriter();
    Message message = new Message(HttpStatus.UNAUTHORIZED, "Error message: " + authEx.getMessage());
    Jackson2JsonEncoder encoder = new Jackson2JsonEncoder();

    writer.write(encoder.getMapper().writeValueAsString(message));
}
```


4.1.2 Controller

Tutti i controller implementati hanno generalmente i seguenti metodi: `getAll()`, `get(id)`, `insert(parametri necessari...)`, `delete(id)`. Essi sono fondamentali per l'utilizzo delle api.

Di seguito ci sono le funzionalità dei metodi per tutti i controller che li implementano: Action, Path, Report, Scheduler e File controller.

Metodo	Tipo di richiesta (HTTP)	Descrizione
getAll()	GET	Ritorna tutti gli attributi di tutti gli oggetti presenti nel database di quel tipo.
get(id)	GET	Ritorna tutti gli attributi l'oggetto che ha come id quello passato come parametro.
insert(parametri...)	PUT	Inserisce un nuovo oggetto nel database
delete(id)	DELETE	Elimina l'oggetto che ha come id quello passato come parametro.

4.1.2.1 PathController

L'attributo `gpr` del `PathController` serve al controller per interfacciarsi con la tabella `GlobalPath` del database. Usa l'annotazione `@Autowired` per indicare a spring che questo parametro dovrà essere creato come Bean e dovrà essere inizializzato alla creazione della classe.

Questo attributo verrà utilizzato in tutti i metodi.

```
@Autowired
GlobalPathRepository gpr;
```

Il metodo **`getAll`** risponde alla richiesta di tipo GET sull'indirizzo **<indirizzo-server>/path** (`localhost:8080/path/`) perché utilizza l'annotazione spring `@GetMapping`.

Risponde ritornando indietro una lista di tutti i `GlobalPath` presenti nel database.

```
@GetMapping()
public @ResponseBody Iterable<GlobalPath> getAll() {
    return gpr.findAll();
}
```


Il metodo **get(path)** risponde alla richiesta GET in egual modo come il metodo *getAll* ma questa volta ritorna solo un *GlobalPath* specifico, quello richiesto dal parametro *path*. Se il parametro *path* è invalido viene ritornato un messaggio d'errore appropriato.

```
@GetMapping(value =("/{path}")
public @ResponseBody Object get(@RequestParam String path) {

    path = path.replaceAll("&#47;", File.separator).trim();
    if (Validator.getPathType(path) != PathType.Invalid) {
        if (gpr.existsById(path))
            return gpr.findById(path).get();
        else {
            return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "Path doesn't exist: " + path);
        }
    } else {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "Invalid path format: " + path);
    }
}
```

Il metodo **insert** risponde alla richiesta di tipo PUT sull'indirizzo **<indirizzo-server>/path** (localhost:8080/path/). I parametri richiesti devono essere presenti nel body della richiesta. Dopo la verifica dei valori dei parametri viene creato un nuovo oggetto di tipo *GlobalPath* e questo viene subito salvato e ritornato come risposta.

```
@PostMapping()
public @ResponseBody Object insert(@RequestParam String path, @RequestParam String ignorePath) {
    path = path.replaceAll("&#47;", File.separator).trim();
    try {
        Path p = Paths.get(path);

        if (!gpr.existsById(p.toAbsolutePath().toString())) {
            if (Validator.getPathType(p.toAbsolutePath().toString()) != PathType.Invalid) {
                if (Files.isReadable(p)) {
                    gpr.save(new GlobalPath(p.toAbsolutePath().toString(), (ignorePath.equals("true"))));
                    return get(path);
                } else {
                    return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "path not readable.: " + path);
                }
            } else {
                return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "Invalid path format: " + path);
            }
        } else {
            return new Message(HttpStatus.BAD_REQUEST, "Path already present in database: " + path);
        }
    } catch (InvalidPathException ipe) {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "Invalid path format: " + path);
    }
}
```

Il metodo ***delete*** risponde alla richiesta di tipo DELETE sull'indirizzo **<indirizzo-server>/path** (localhost:8080/path/). Come parametro è richiesto solo l'id come nel metodo *get* dopo che viene fatta una verifica del parametro passato l'Oggetto *GlobalPath*, se esiste nel database, viene eliminato e l'oggetto eliminato viene ritornato come risposta.

```
@DeleteMapping()
public @ResponseBody Object delete(@RequestParam String path) {

    PathType type = Validator.getPathType(path);
    path = path.replaceAll("&#47;", File.separator).trim();

    if (gpr.existsById(path) && type != PathType.Invalid) {
        GlobalPath entry = gpr.findById(path).get();
        gpr.delete(entry);
        return entry;
    } else {
        return new Message(HttpStatus.BAD_REQUEST, "Invalid path: " + path);
    }
}
```

Questi metodi in termini di funzionamento sono uguali nei controller PathController, FileController, ReportController, SchedulerController e ActionController.

Di seguito saranno descritti i metodi particolari di delle classi:

4.1.2.2 ActionController

L'attributo *context* rappresenta il contest dell'applicazione. Può essere usato per creare o eliminare delle classi dell'applicazione.

```
@Autowired
private ApplicationContext context;
```

Il metodo `checkPath` risponde alla richiesta di tipo POST sull'indirizzo **<indirizzo-server>/action/path/** (localhost:8080/action/path). Il metodo controlla se il path passato come parametro è valido oppure no. Il metodo viene usato dalla *GUI* per verificare la validità di un percorso quando l'utente sceglie di muovere un duplicato in una nuova posizione. Il percorso passato deve essere una cartella che si trova sul disco per essere considerato come valido.

```
@PostMapping("/path")
public @ResponseBody Message checkPath(@RequestParam String path) {
    return new Message(HttpStatus.OK, String.valueOf(Validator.getPathType(path).equals(PathType.Directory)));
}
```

Il metodo `executeActions` viene chiamato quando viene fatta una richiesta di tipo POST sull'indirizzo **<indirizzo-server>/action/execute/all** (localhost:8080/action/execute/all)

Questo metodo esegue immediatamente tutte le azioni dal database che non sono ancora state eseguite. Viene creato un nuovo `actionsManager` e tramite l'attributo `context` e poi viene fatto partire.

```
@PostMapping("/execute/all")
public @ResponseBody Object executeActions() {

    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String authenticatedUser = authentication.getName();

    AuthenticationDetails internalUser = adr.findById(authenticatedUser).get();

    BeanDefinitionRegistry factory = (BeanDefinitionRegistry) context.getAutowireCapableBeanFactory();
    ((DefaultListableBeanFactory) factory).destroySingleton("actionsManager");

    ActionsManager manager = (ActionsManager) context.getBean("actionsManager");
    manager.setActions(actionRepository.findActionsFromUser(internalUser));
    manager.setUser(internalUser);
    ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadPool(1);
    scheduledExecutorService.schedule(manager, 0L, TimeUnit.SECONDS);

    return actionRepository.findActionsFromUser(internalUser);
}
```

4.1.2.3 SchedulerController

Il metodo `getFirstPosition` ritorna la posizione del primo bit a 1 in un intero. Questo metodo viene usato per prendere la data d'esecuzione di una scansione pianificata.

```
public Integer getFirstPosition(Integer number, int max) {
    List<Integer> positions = this.getPositions(number, max);
    if (positions.size() > 0) {
        return positions.get(0);
    }
    return 0;
}
```

Il metodo `getFirstPosition` ritorna una lista di posizioni dei bit a 1 in un intero. Questo metodo viene usato dal metodo `getFirstPosition` per prendere la data d'esecuzione di uno scheduler.

Ritorna una lista di posizioni che in futuro possono essere rappresentati come giorni del mese nei quali eseguire le scansioni.

```
public List<Integer> getPositions(Integer number, int max) {

    max = max < 4 ? max = 31 : max;

    List<Integer> positions = new ArrayList<>();

    for (int i = max; i >= 0; i--) {
        if ((number & (1 << i)) != 0) {
            positions.add(i);
        }
    }
    return positions;
}
```

4.1.2.4 LoginController

Il `LoginController` non implementa i metodi `get`, `getAll` perché non sono necessari. Questo controller viene usato per controllare se un utente è autenticato grazie al metodo `checkLogin`

```
@RequestMapping("/")
public @ResponseBody Message checkLogin() {
    return new Message(HttpStatus.OK, "User authenticated successfully");
}
```

Se come risposta il client riceve il messaggio *"User authenticated successfully"* vuol dire che l'autenticazione è giusta mentre se riceve un messaggio d'errore generato da spring vuol dire che le credenziali sono sbagliate.

Il `LoginController` per mette anche l'inserimento di nuovi utenti e l'eliminazione

4.1.2.5 ScanController

Il ScanController non implementa i metodi insert, delete, get e getAll perché questa classe non viene usata per modificare i dati ma per controllare il servizio stesso.

Il metodo start viene chiamato quando l'utente fa una richiesta di tipo POST sull'indirizzo **<indirizzo-server>/scan/start/** (localhost:8080/scan/start/). Il metodo start avvia una nuova scannerizzazione che va a controllare i file che si trovano sotto i percorsi definiti nel database nella tabella GlobalPath grazie al controller PathController.

```
@PostMapping("/start")
public @ResponseBody Object start(@RequestParam(required = false) Integer threadCount) {

    if (gpr.count() > 0) {

        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        String authenticatedUser = authentication.getName();
        AuthenticationDetails internalUser = adr.findById(authenticatedUser).get();

        currentScan = (ScanManager) context.getBean("scanManager");
        report = new Report(internalUser);
        report.setStart(System.currentTimeMillis());
        reportRepository.save(report);

        currentScan.setReportRepository(reportRepository);
        currentScan.setReportId(report.getId());
        currentScan.setThreadCount(threadCount);
        currentScan.setListener(this);
        currentScan.start();
        return report;
    } else {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "No path to scan set");
    }
}
```

Il metodo stop viene chiamato quando l'utente fa una richiesta di tipo POST sull'indirizzo **<indirizzo-server>/scan/stop/** (localhost:8080/scan/stop/). Il metodo stop interrompe la scansione e distrugge gli oggetti relativi a quella scansione.

```

@PostMapping("/stop")
public @ResponseBody Object stop() {
    if (currentScan != null) {
        currentScan.stopScan();

        System.out.println("Waiting finish");
        while (currentScan.isAlive()) {
            long time = System.currentTimeMillis();
            if (System.currentTimeMillis() - time > 100) {
                System.out.print(".");
            }
        }
        Report report = currentScan.getReport();
        destroyScanManager();
        return report;
    } else {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "No scan currently running");
    }
}

```

Il metodo pause viene chiamato quando l'utente fa una richiesta di tipo POST sull'indirizzo **<indirizzo-server>/scan/pause/** (localhost:8080/scan/pause/). Il metodo pause ferma la scansione mantenendo la possibilità di riprendere l'esecuzione in un secondo momento.

```
@PostMapping("/pause")
public @ResponseBody Message pause() {
    if (currentScan != null) {
        if (!currentScan.isPaused()) {
            currentScan.pauseAll();
            return new Message(HttpStatus.OK, "Scan paused");
        } else {
            return new Message(HttpStatus.OK, "Scan already paused");
        }
    } else {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "No scan currently runnin");
    }
}
```

Il metodo resume viene chiamato quando l'utente fa una richiesta di tipo POST sull'indirizzo **<indirizzo-server>/scan/resume/** (localhost:8080/scan/resume/). Il metodo resume prosegue con l'esecuzione della scansione dallo stato di pausa.

```
@PostMapping("/resume")
public @ResponseBody Message resume() {
    if (currentScan != null) {
        if (currentScan.isPaused())
            currentScan.resumeAll();
        return new Message(HttpStatus.OK, "Scan resumed");
    } else {
        return new Message(HttpStatus.INTERNAL_SERVER_ERROR, "No scan currently running");
    }
}
```

Il metodo getStatus viene chiamato quando l'utente fa una richiesta di tipo GET sull'indirizzo **<indirizzo-server>/scan/status/** (localhost:8080/scan/status/). Il metodo getStatus restituisce dei dati sulla scansione. I dati restituiti sono: i file scansionati, la durata e se è in corso oppure no.

```
@GetMapping("/status")
public @ResponseBody Object getStatus() {
    int count = report.getFilesScanned() == null ? 0 : report.getFilesScanned();
    Message response;
    if(report != null){
        response = new Message(HttpStatus.OK, String.valueOf(count));
    }else{
        response = new Message(HttpStatus.NOT_FOUND, String.valueOf(count));
    }
    LocalDateTime time = Instant.ofEpochMilli(report.getStart() == null ? 0 : report.getStart())
        .atZone(ZoneId.systemDefault()).toLocalDateTime();
    response.setTimestamp(time);
    return response;
}
```

Il metodo `destroyScanManager` distrugge l'attributo `currentScan` grazie all'attributo `context` per permettere un nuovo avvio della scansione con dei oggetti nuovi.

```
private void destroyScanManager(){
    BeanDefinitionRegistry factory = (BeanDefinitionRegistry) context.getAutowireCapableBeanFactory();
    ((DefaultListableBeanFactory) factory).destroySingleton("scanManager");
}
```

4.1.3 Entity

Il framework Spring usa java Hibernate per eseguire le operazioni ORM. Di seguito spiegherò alcune annotazioni usate sui attributi delle classi.

Il package `entity` nel progetto `deduplicato` contiene le entità del database, cioè le definizioni delle tabelle del database.

Tutte le classi usano codice standard java. Gli attributi degli oggetti diventeranno le colonne nel database.

Ogni classe usa l'annotazione `@Entity` per indicare a spring che la classe descrive una tabella e che quella tabella dovrà essere creata all'avvio dell'applicazione.

Alcuni attributi contengono l'annotazione `@Id` `@Id` per indicare a spring che quel attributo sarà la chiave primaria della tabella.

Come in MySQL ogni classe deve avere almeno un attributo con l'annotazione `@Id`

Alcuni attributi usano anche l'annotazione `@GeneratedValue` per indicare a spring che il valore di quel attributo dovrà essere generato automaticamente, è comparabile al **auto_increment** di **MYSQL** che automaticamente incrementa il valore del id intero.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
```

Esistono dei attributi opzionali e quelli utilizzano l'annotazione `@Nullable` `@Nullable` per indicare a spring che il valore in quella colonna poterbbe essere nullo.


Le colonne con relazioni molti a uno usano l'annotazione

```
@ManyToOne(fetch = FetchType.EAGER)
```

In più definiscono anche il modo in cui verranno presi i dati impostando il `FetchType`. `FetchType.EAGER` significa che i valori associati a quella colonna dovranno essere caricati subito alla richiesta del primo dato, mentre la `FetchType.LAZY` cerca di ritardare il recupero dei dati al più tardi possibile, nel frattempo il campo rimane vuoto.

4.1.4 Repository

Il package `repository` contiene le interfacce `repository` che estendono l'interfaccia `CrudRepository`

	SAMT - Sezione Informatica	Pagina 41 di 64
	Deduplicatore di files	

Tutte i repository permettono di fare operazioni CRUD sul database sulla tabella definita in base al tipo di repository. Nei repository si possono definire query MYSQL personalizzate usando l'annotazione @Query

Il repository DuplicateRepository che cerca i file scansionati per trovare i duplicati usa una query personalizzata grazie all'annotazione @Query, usa l'opzione nativeQuery = true per permettere di ritornare una lista di tipo java.lang.List di Duplicates. Questo repository è quello più importante.

```
@Repository
public interface DuplicateRepository extends CrudRepository<Duplicate, String> {
    @Query( value = "SELECT f.size,f.hash,count(*) as count "+
        "FROM file f "+
        "WHERE report=?1 "+
        "GROUP BY f.hash,f.size "+
        "HAVING count > 1 "+
        "ORDER BY hash,size DESC",
        nativeQuery = true)
    List<Duplicate> findDuplicatesFromReport(Report report);
}
```

4.1.5 Scanner

Il package Scanner contiene tutto quello che riguarda la scansione.

4.1.5.1 ScanManager

La classe ScanManager gestisce le thread di scansione.

All'avvio dello ScanManager vengono eseguiti i seguenti passi:
Viene impostato il numero massimo di thread da eseguire (default 10)
vengono caricati i percorsi da scannerizzare e quelli da ignorare.

```
pool = Executors.newFixedThreadPool(threadCount);
report = getReport();

paths = gpr.findAll().iterator();

List<GlobalPath> ignorePathsFromRepository = gpr.findIgnored();

List<String> ignorePaths = new ArrayList<>();
List<String> ignores = new ArrayList<>();

for (GlobalPath ignorePath : ignorePathsFromRepository) {
    if (ignorePath.isFile())
        ignores.add(ignorePath.getPath());
    else
        ignorePaths.add(ignorePath.getPath());
}
```

Subito dopo viene fatto partire un nuovo ScannerThread per ogni percorso che si trova nel database e poi si aspetta la fine d'esecuzione delle thread e le loro sotto-thread. Per ogni ScannerThread viene passato un oggetto monitor che verrà utilizzato per riprendere l'esecuzione delle thread una volta fermate.

```
while (paths.hasNext()) {
    ScannerThread thread = new ScannerThread(Paths.get(paths.next().getPath()), this, report,
        fileRepository, ignorePaths, ignores, monitor);
    rootThreads.add(thread);
    pool.execute(thread);
}

pool.shutdown();
pool.awaitTermination(terminationTimeout, TimeUnit.SECONDS);
```

Alla fine dell'esecuzione viene salvato aggiornato lo stato del rapporto e vengono salvati i dati del rapporto sul database.

Infine viene notificato il listener (ScanController) che la scansione sia finita per eseguire la pulizia dell'oggetto ScanManager

```
} finally {
    pool.shutdownNow();

    List<Duplicate> duplicates = duplicateRepository.findDuplicatesFromReport(report);

    report.setAverageDuplicateCount((float) duplicates.size() / (float) filesScanned);
    report.setDuration((System.currentTimeMillis() - report.getStart()));
    reportRepository.save(report);

    System.out.println("[INFO] Scan manager Finished");
    if (listener != null)
        listener.scanFinished();
}
```

4.1.5.2 ScannerThread

La classe ScannerThread esegue l'effettiva scansione delle cartelle.

Per prima cosa elenca tutti gli elementi contenuti nel percorso impostato da scansionare

Poi esegue il controllo se sono file o cartella.

Se l'elemento è un file lo aggiunge alla lista dei file trovati altrimenti fa partire un nuovo ScannerThread che andrà a controllare i contenuti di quella cartella, facendo così il filesystem viene scannerizzato in modo recursivo. Le thread avviate rimangono comunque 10 per via del Pool di thread definito nel ScanManager.

```
public void run() {
    LinkedList<File> files = new LinkedList<File>();
    try {
        if (!ignoreFound) {

            File[] list = new File(rootPath.toString()).listFiles();

            for (File file : list) {
                System.out.println("[INFO] Scanning directory: " + file.getAbsolutePath());
                if (!Thread.interrupted()) {
                    checkPaused();
                    if (file.isFile()) {
                        if (!ignores.contains(file.getAbsolutePath())) {
                            files.add(file);
                        } else {
                            System.out.println("[INFO] File not saved, set to ignore: " + file.getAbsolutePath());
                        }
                    } else if (file.isDirectory()) {
                        ScannerThread thread = new ScannerThread(Paths.get(file.getAbsolutePath()), listener,
                            report, fileRepository, ignorePaths, ignores, monitor);
                        children.add(thread);
                    }
                }
            }
        }
    } finally {

        if (files.size() > 0) {
            while (files.peek() != null) {

                Hasher hasher = new Hasher(files.poll(), report, this, fileRepository, monitor);
                hashers.add(hasher);
                pool.execute(hasher);
            }
            pool.shutdown();
            try {
                pool.awaitTermination(DEFAULT_TERMINATION_TIMEOUT, TimeUnit.SECONDS);
            } catch (InterruptedException ie) {
                System.err.println("[ERROR] Thread interrupted: " + ie.getStackTrace().toString());
                pool.shutdownNow();
            } finally {
                pool.shutdownNow();
            }
        }
        listener.addFilesScanned(filesScanned);
    }
}
```

4.1.5.3 Hasher

La classe Hasher si occupa di generare una hash di un file trovato e di aggiungerlo al repository

```
Long lastModified = file.lastModified();
try {
    RandomAccessFile fileRAF = new RandomAccessFile(file.getAbsolutePath(), "r");
    String hash = getHash(fileRAF, "MD5");
    long size = fileRAF.length();

    fileRAF.close();

    File record = new File(file.getAbsolutePath(), lastModified, hash, size, report);
    fileRepository.save(record);
    stl.addFilesScanned(0);
}
```

Il metodo principale è il metodo **getHash** che genera una hash di tipo MD5 dei contenuti del file leggendolo a blocchi di 32 KB, chunk di grandezza più piccola riducevano troppo le prestazioni.

```
private final static int BUFFER_SIZE = 32768;

byte[] buffer = new byte[BUFFER_SIZE];
long read = 0;

long end = file.length();
int unitSize;

while (read < end) {
    checkPaused();
    unitSize = (int) (((end - read) >= BUFFER_SIZE) ? BUFFER_SIZE : (end - read)); // controllo se sono
                                                                                     // arrivato in fondo al
                                                                                     // file.
    file.read(buffer, 0, unitSize); // leggo un chunk del file definito dall'attributo BUFFER_SIZE
    messageDigest.update(buffer, 0, unitSize); // aggiorno il hash con i nuovi dati letti.
    read += unitSize; // sposto il buffer al prossimo chunk di dati.
}
```

4.1.6 Validator

La classe Validator contiene diversi metodi statici, utili per eseguire dei controlli sui dati in arrivo nei metodi dei controller.

Il metodo getPathType controlla la validità e il tipo di percorso passato come parametro.

```
public static PathType getPathType(String path) {
    try {
        if (path != null) {

            String pathParsed = path.replaceAll("&#47;", File.separator);
            Path pa = Paths.get(pathParsed);

            if (Files.isDirectory(pa)) {
                return PathType.Directory;
            } else {
                if (Files.exists(pa)) {
                    return PathType.File;
                } else {
                    return PathType.Invalid;
                }
            }
        } else {
            return PathType.Invalid;
        }
    } catch (InvalidPathException | NullPointerException | SecurityException ex) {
        System.err.println("[ERROR] " + ex.getMessage());
        return PathType.Invalid;
    }
}
```


Il metodo isInt e isLong eseguono un semplice cast per verificare che l'input in formato stringa sia numerico di tipo int o long rispettivamente.

```
public static Integer isInt(String input) {
    try {
        return Integer.parseInt(input);
    } catch (NumberFormatException nfe) {
        return null;
    }
}

public static Long isLong(String input) {
    try {
        return Long.parseLong(input);
    } catch (NumberFormatException nfe) {
        return null;
    }
}
```

Il metodo isHex controlla se una stringa sia in formato esadecimale oppure no.

```
public static String isHex(String input) {
    return Pattern.matches("[0-9a-fA-F]+", input) ? input : null;
}
```

Il metodo getActionType tenta di identificare il tipo del enum ActionType in base alla stringa passata come parametro.

```
public static String getActionType(String type) {
    if (type.equalsIgnoreCase(ActionType.DELETE) || type.equalsIgnoreCase(ActionType.MOVE)
        || type.equalsIgnoreCase(ActionType.IGNORE) || type.equalsIgnoreCase(ActionType.SCAN))
        return type.toUpperCase();
    else
        return null;
}
```

4.1.7 Resources

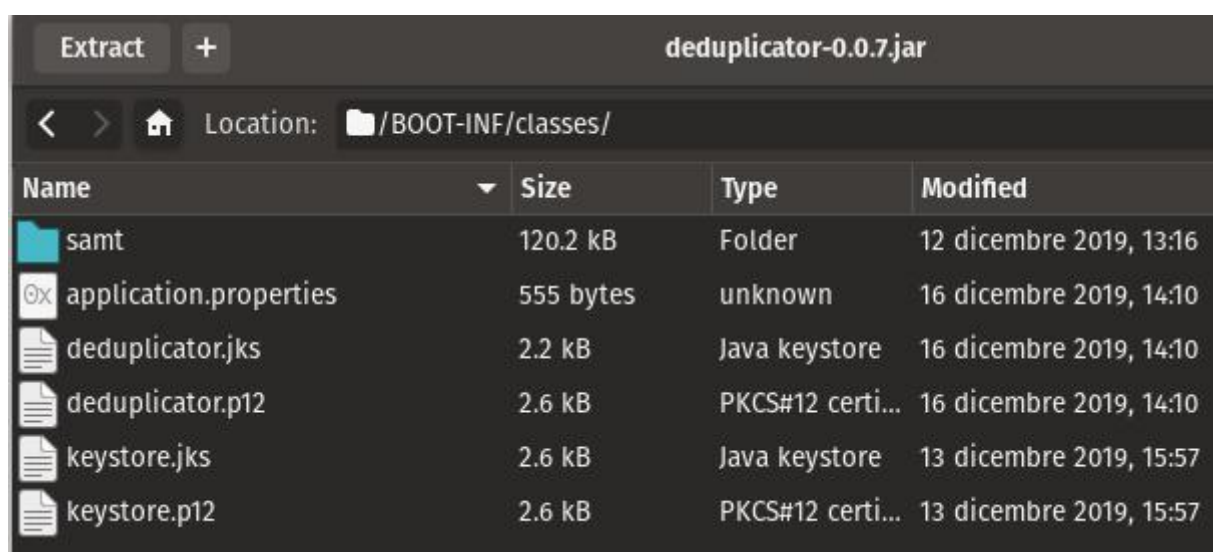
La cartella resources all'interno del progetto contiene un file *application.properties* che contiene diverse impostazioni riguardo il progetto come la stringa di connessione per il database, le credenziali del database, la porta sul quale girerà il servizio, la posizione della chiave privata per la comunicazione https, la password della chiave per la comunicazione e il tipo della chiave.

```

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/
deduplicator?createDatabaseIfNotExist=true&useUnicode=true&
useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&
serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=Password&1
javax.persistence.create-database-schemas=true
server.port=8443
server.ssl.enabled=true
server.ssl.key-store-type=PKCS12
server.ssl.key-store=classpath:deduplicator.p12
server.ssl.key-store-password=Password&1
server.ssl.key-alias=deduplicator

```

La configurazione del progetto può essere modificata aprendo il file .jar come un archivio, navigando verso **/BOOT-INF/classes/** e modificando il file *application.properties*. Questa modifica è utile se si vuole per cambiare la chiave per la comunicazione HTTPS oppure cambiare le credenziali e/o stringa di accesso al database.



4.2 DeduplicatorGUI

Il secondo pezzo del progetto si chiama DeduplicatorGUI. Esso offre all'utente un'interfaccia grafica per controllare il servizio **deduplicator**.

4.2.1 Communication

Il package communication contiene l'unica classe Client che si occupa di mandare e ricevere le richieste alle REST API.

Le gui sono state create in swing grazie al editor drag and drop di netbeans.

4.2.1.1 Client

La classe Client viene usata dai pannelli della GUI per eseguire determinate opzioni richieste dal utente.

L'attributo CA_PASS definisce la password del certificato HTTPS, viene usato nel costruttore per impostare il certificato nel KeyStore.

```
private final String CA_PASS = "Password&1";
private KeyStore keyStore;
```

Il certificato viene caricato dalla posizione "../resource/deduplicato.p12" relativa a quella della posizione della classe Client. Il file viene letto in formato InputStream.

```
InputStream in = getClass().getResourceAsStream("../resource/deduplicator.p12");
try{
    KeyStore keyStore = KeyStore.getInstance("PKCS12");
    keyStore.load(in, CA_PASS.toCharArray());
}
```

Infine il certificato viene inserito nell'attributo restTemplate che è quello usato per fare le richieste al servizio.

```
HttpComponentsClientHttpRequestFactory requestFactory = null;
try {
    TrustStrategy acceptingTrustStrategy = (X509Certificate[] chain, String authType)
-> true;
    SSLContext sslContext =
    SSLContextBuilder.create()
    .loadKeyMaterial(keyStore, CA_PASS.toCharArray())
    .loadTrustMaterial(null, acceptingTrustStrategy)
    .build();

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

    requestFactory = new HttpComponentsClientHttpRequestFactory();

    requestFactory.setHttpClient(httpClient);
} catch (UnrecoverableKeyException | NoSuchAlgorithmException | KeyStoreException |
KeyManagementException e) {

    System.out.println("Unable to create client: " + e.getMessage());
    e.printStackTrace();
}
if (requestFactory != null)
    restTemplate = new RestTemplate(requestFactory);
```

Il metodo `get` effettua una richiesta di tipo `get` sul path impostato come parametro e restituisce il messaggio ricevuto in formato `ResponseEntity` di `spring`.

```
public ResponseEntity<String> get(String path) {

    HttpEntity<Map<String, String>> requestEntity = new HttpEntity<>(createHeaders(false))
    ;
    try {
        ResponseEntity<String> response = restTemplate.exchange(
            "https://" + host + ":" + port + "/" + path, HttpMethod.GET,
            requestEntity,
            String.class);

        if (response.getStatusCode().equals(HttpStatus.OK)) {

            return response;

        } else {
            return null;
        }
    } catch (RestClientException rce) {
        System.out.println("[ERROR] Rest client exception: " + rce);
    }
    return null;
}
```

Il metodo `delete` effettua una richiesta di tipo `delete` sull'indirizzo impostato come parametro. Il metodo `delete` viene usato solo dalla schermata `Path` perciò la chiave del parametro del body predefinito è impostato come `"path"`. Il parametro del metodo `param` contiene il percorso da eliminare.

```
public ResponseEntity<String> delete(String path, String param) {

    MultiValueMap<String, Object> values = new LinkedMultiValueMap<>();
    values.add("path", param);

    HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(values,
        createHeaders(true));

    ResponseEntity<String> response = null;
    try {
        response = restTemplate.exchange("https://" + host + ":" + port + "/path/",
            HttpMethod.DELETE, requestEntity, String.class);
    } catch (RestClientException rce) {
        System.out.println("rce: " + rce.getMessage());
    }

    if (response != null) {
        return response;
    } else {
        return new ResponseEntity<String>(HttpStatus.BAD_REQUEST);
    }
}
```

Il metodo `createHeaders` crea il header di ogni richiesta, viene impostato il parametro del header che contiene l'autenticazione e charset.

```
private HttpHeaders createHeaders(boolean hasFormData) {
    HttpHeaders header = new HttpHeaders();

    String auth = username + ":" + password;
    byte[] encodedAuth = Base64.getEncoder().encode(auth.getBytes(
        Charset.forName("US-ASCII")));
    String authHeader = "Basic " + new String(encodedAuth);

    header.add("Authorization", authHeader);

    if (hasFormData) {
        header.setContentType(MediaType.MULTIPART_FORM_DATA);
    }
    return header;
}
```

Il metodo `post` manda una richiesta di tipo POST al percorso impostato come parametro del metodo e imposta i valori della parte del body della richiesta grazie al parametro `values` di tipo `MultiValueMap`.

```
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("https://" + host + ":" +
            port + "/" + path,
            HttpMethod.POST, requestEntity, String.class);
    } catch (RestClientException rce) {
        System.out.println("rce: " + rce.getMessage());
    }
    return response;
}
```

Il metodo put manda una richiesta di tipo PUT all'indirizzo impostato come parametro. Come il metodo post, imposta i valori della body della richiesta grazie al parametro *values* di tipo MultiValueMap.

```
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("https://" + host + ":" +
            port + "/" + path,
            HttpMethod.PUT, requestEntity, String.class);
    } catch (RestClientException rce) {
        System.out.println("Rest client exception: ");
        StackTraceElement[] st = rce.getStackTrace();
        for (StackTraceElement stackTraceElement : st) {
            System.out.println(stackTraceElement.toString());
        }
    }

    return response;
}
```


La classe BaseJPanel definisce il pannello di base che verrà implementato da tutte le altre schermate della GUI.

Ogni schermata della gui avrà un riferimento al client per eseguire le richieste e il metodo getArray che trasforma un oggetto di tipo JSONArray in uno di tipo JSONObject[].

```
public class BaseJPanel extends JPanel{

    /**
     * Il client per le richieste.
     * È di tipo {@link Client}.
     */
    private Client client;
    JSONParser parser = new JSONParser();

    /**
     * @param client il client da impostare.
     */
    public void setClient(Client client) {
        this.client = client;
    }

    /**
     * @return l'oggetto client.
     */
    public Client getClient() {
        return client;
    }

    /**
     * Metodo che sarà sovrascritto dagli altri panel del progetto.
     */
    public void tabSelected(){

    }

    /**
     * Il metodo getArray trasforma un oggetto di tipo JSONArray in JSONObject[].
     * @param array l'array da trasformare.
     * @return l'array di JSONObject.
     */
    public JSONObject[] getArray(JSONArray array) {

        Object[] objectArray = (Object[]) array.toArray();
        JSONParser parser = new JSONParser();
        JSONObject[] result = new JSONObject[objectArray.length];
        for (int i = 0; i < objectArray.length; i++) {
            try {
                result[i] = (JSONObject) parser.parse(objectArray[i].toString());
            } catch (ParseException e) {
                System.out.println("unable to parse " + objectArray[i].toString());
            }
        }
        return result;
    }
}
```

4.2.2 MainJFrame

La classe MainJFrame contiene un JTabbedPane che offre la possibilità di cambiare la visuale delle varie schermate tramite i tab in alto



Alla selezione di una nuova tab viene chiamato il metodo tabSelected di quella schermata.

```
@Override
public void stateChanged(ChangeEvent e) {
    ( (BaseJPanel)menu.getSelectedComponent()).tabSelected();
}
```

Il metodo tabSelected in tutte le schermate viene usato per caricare i dati necessari. Come nella schermata paths, quando essa viene aperta vengono subito caricati tutti i percorsi presenti nel database del servizio.

```
@Override
public void tabSelected() {
    updatePathsList();
}

private void updatePathsList() {
    ResponseEntity<String> response = getClient().get("path/");

    if (response != null && response.getStatusCode().equals(HttpStatus.OK)) {
        try {
            JSONObject[] array = getArray((JSONArray) parser.parse(response.getBody()));
            pathJList.setModel(new AbstractListModel<String>() {
                public int getSize() {
                    return array.length;
                }

                public String getElementAt(int i) {
                    return array[i].get("path").toString();
                }
            });

            typeJList.setModel(new AbstractListModel<String>() {
                public int getSize() {
                    return array.length;
                }

                public String getElementAt(int i) {
                    return array[i].get("ignoreFile").toString();
                }
            });
        } catch (ParseException pe) {
            JOptionPane.showMessageDialog(this, "Unable to get retrieve paths: " + pe.getMessage(), "Get error ",
                JOptionPane.INFORMATION_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(this, "Unable to get retrieve paths", "Get error ",
            JOptionPane.INFORMATION_MESSAGE);
    }

    pathJScrollPane.revalidate();
    pathJScrollPane.repaint();
}
```

Il metodo `userConnected` viene usato dalla classe `MainJFrame` per abilitare l'uso di tutte le chermate una volta che l'utente inserisce delle credenziali valide nella schermata login.

```
@Override
public void userConnected(Client client) {
    changeTabsAccessibility(menu, true);
    for (int i = 1; i < menu.getTabCount(); i++) {
        ((BaseJPanel) menu.getComponentAt(i)).setClient(client);
    }
}

private void changeTabsAccessibility(JTabbedPane menu, boolean state) {
    for (int i = 1; i < menu.getTabCount(); i++) {
        menu.setEnabledAt(i, state);
    }
}
```

4.2.2.1 DuplicatesComboBoxMode

Questa classe estende il `DefaultComboBoxModel` che viene usato nella schermata `Duplicates` per mostrare la lista dei file quando si sceglie un duplicato.

Il metodo `getElementAt` viene usato dal `JScrollPane` per ricavare il valore del testo da stampare su una riga.

```
public String getElementAt(int i) {
    return "Id: " + i + " Count: " + array[i].get("count").toString() + " Size: " + getSize(Double.valueOf(
        array[i].get("size").toString()));
}

private String getSize(Double sizeDouble) {
    String size = "";
    DecimalFormat formatter = new DecimalFormat("0.0##");
    if (sizeDouble > 1073741824.0)
        size = formatter.format(sizeDouble / 1073741824.0) + " GB";
    else if (sizeDouble > 1048576.0)
        size = formatter.format(sizeDouble / 1048576.0) + " MB";
    else if (sizeDouble > 1024.0)
        size = formatter.format(sizeDouble / 1024.0) + " KB";
    else
        size = sizeDouble + " B";
    return size;
}
```

Per stampare la grandezza del file viene usato il metodo `getSize` che formatta il numero di bytes in GB,MB,KB oppure B in base alla grandezza del file.

5 Test

5.1 Protocollo di test

Tutti i test sono stati eseguiti tramite il client GUI del progetto.

Test Case: Riferimento:	TC-001 REQ-003	Nome:	La creazione dei rapporti.
Descrizione:	Far partire una scansione e controllare la generazione del rapporto per quella scansione.		
Prerequisiti:	Il servizio deduplicator è in esecuzione Eseguire il login tramite la GUI. Impostare un percorso da scansionare. La scansione sia finita		
Procedura:	1. Selezionare il tab duplicates 2. Selezionare un rapporto dal dropdown menu che si trova in alto a sinistra 3. Selezionare un duplicato (se ci sono)		
Risultati attesi:	Una lista di file sotto i dropdown menu sui quali è possibile impostare un'azione da eseguire.		

Test Case: Riferimento:	TC-002 REQ-004	Nome:	L'esecuzione delle azioni impostate
Descrizione:	Verificare l'esecuzione delle azioni impostate alla revisione di un rapporto di una scansione.		
Prerequisiti:	Il servizio deduplicator è in esecuzione Esiste un rapporto		
Procedura:	<ol style="list-style-type: none"> 1. Selezionare il tab duplicates 2. Selezionare un rapporto dal dropdown menu che si trova in alto a sinistra 3. Selezionare un duplicato 4. Selezionare un file dalla lista 5. Scegliere l'opzione elimina 6. Impostare la data e ora d'esecuzione 7. Schiacciare il tasto apply 8. Aspettare la data e ora d'esecuzione e poi rifare il scan 9. Controllare la presenza del file eliminato nel nuovo scan 		
Risultati attesi:	Il file non deve essere più presente nella lista dei duplicati perché sarà stato eliminato dall'azione impostata.		

Test Case: Riferimento:	TC-003 REQ-006	Nome:	Messa in pausa della scansione
Descrizione:	Avere la possibilità fermare l'esecuzione di una scansione in corso.		
Prerequisiti:	<p>Il servizio deduplicator è in esecuzione</p> <p>Almeno un percorso è inserito per la scansione</p>		
Procedura:	<ol style="list-style-type: none"> 1. Cliccare il tab scans 2. Avviare una scansione schiacciando il tasto Start Scan 3. Dopo un paio di secondi schiacciare il tasto Pause Scan 4. Controllare sull'output del server che l'esecuzione si è fermata 5. Schiacciare il tasto Resume Scan per riprendere la scansione 		
Risultati attesi:	Tutte le thread di scansione si fermano quando viene schiacciato il tasto Pause Scan, Schiacciando il tasto Resume Scan tutte le thread di scansione riprendono con la scansione.		

Test Case: Riferimento:	TC-004 REQ-007	Nome:	Gestione dei rapporti
Descrizione:	La possibilità di vedere i rapporti passati		
Prerequisiti:	<p>Il servizio è in esecuzione</p> <p>Eseguire il login tramite la GUI.</p> <p>Eseguire un paio di scansioni</p>		
Procedura:	<ol style="list-style-type: none"> 1. Selezionare il tab Duplicates e verificare che tutti i rapporti sono presenti nel dropdown menu 2. Selezionare i rapporti uno a uno e verificare i dati con il tasto info. 		
Risultati attesi:	Tutti i rapporti sono presenti e navigabili tramite il dropdown menu. I duplicati che si trovano in un rapporto vecchio, verranno spostati automaticamente sul rapporto più recente per avere le informazioni sui duplicati più recenti.		

Test Case: Riferimento:	TC-005 REQ-008	Nome:	Scheduler delle scansioni
Descrizione:	<p>La possibilità di impostare una scansione pianificata.</p> <p>Questo test è da eseguire con Postman oppure un altro tool per fare richieste HTTP/HTTPS perché non è stata implementata questa funzione nella GUI.</p>		
Prerequisiti:	<p>Il servizio è in esecuzione</p> <p>I certificati sono impostati in Postman</p>		
Procedura:	<p>Fare una richiesta PUT sull'indirizzo <ip server>/scheduler/ con i parametri nel body (form-data)</p> <p>monthly:null weekly:null timeStart:<data e ora d'inizio in formato timestamp> (possibilmente 5 min dall'ora attuale) repeated:false</p> <p>inviare la richiesta e verificare che la risposta sia una con il header 200 OK</p>		
Risultati attesi:	<p>Osservare l'output nel server e dopo 5 min dall'invio della richiesta dovrebbe partire una nuova scansione.</p>		

5.2 Risultati test

Requisito	Soddisfatto o Si/No	Note
REQ-01	Si	Autenticazione BASIC + HTTPS con certificato self-signed
REQ-02	Si	
REQ-03	Si	
REQ-04	No	Le azioni da eseguire vengono aggiunte ma non vengono eseguite quando arriva il tempo di eseguirle
REQ-05	Si	GUI fatta in java swing nativo
REQ-06	Si	
REQ-07	Si	I rapporti vengono salvati e si possono vedere i vecchi rapporti, ma la continuazione di un rapporto non finito non è stato implementato
REQ-08	No	Come il requisito REQ-04, le scansioni si possono aggiungere ma non vengono eseguite.
REQ-09	No	Il rilevamento in tempo reale grazie ai trigger di sistema, è un requisito opzionale

TEST	Riuscito Si/No	Note
TC-001	Si	
TC-002	No	
TC-003	Si	
TC-004	Si	
TC-005	No	

5.3 Mancanze/limitazioni conosciute

Il motivo principale di queste mancanze/limitazioni è la mancanza di tempo a disposizione. Il progetto è semplicemente troppo lungo da implementare per una persona singola sia la parte REST api sia la grafica e in più eseguire la verifica che tutto funzioni una volta messe assieme le due parti.

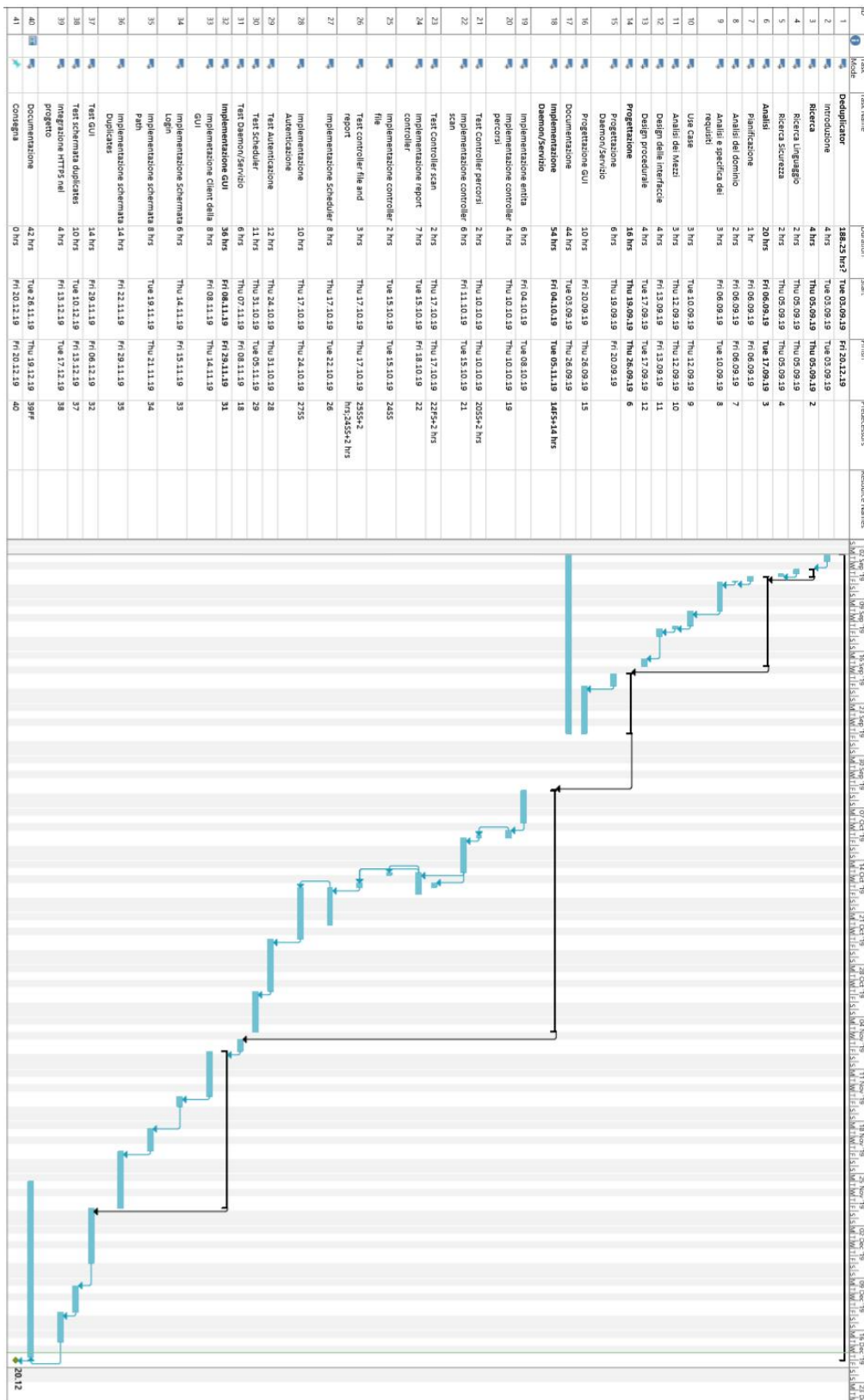
Non è stata implementata la parte dell'inserimento delle scansioni pianificate (tab scheduler) tramite la GUI anche se la parte server c'è.

Le azioni da eseguire sui duplicati vengono mandate al server ma questo non le riconosce e quindi non le aggiunge alla tabella Actions.

Non c'è un modo per aggiornare lo stato della scansione tramite la GUI senza provocare problemi di concorrenza e accesso multiplo ai dati per via dell'utilizzo di mysql, servirebbe cambiare il sistema di scansione tenendo conto di poter ricavare lo stato della scansione a ogni momento.

Non si può riprendere l'esecuzione di scansioni vecchie o non complete ma solo di quella ultima.

6 Consuntivo



L'immagine del gantt consuntivo verrà messa in allegato in formato più grande.

7 Conclusioni

Il progetto è molto interessante e mi è piaciuto tanto soprattutto perchè ho avuto la possibilità di imparare un nuovo framework e di creare qualcosa che è utile e si può usare nella vita reale. Se avessi avuto il tempo di finirlo lo avrei usato per uso personale.

7.1 Sviluppi futuri

Come sviluppo futuro per prima cosa sarebbe di finire il progetto con tutte le funzionalità richieste. Sarebbe bello anche creare una GUI più user friendly per esempio come sito web.

7.2 Considerazioni personali

Cosa ho imparato in questo progetto? ecc

8 Sitografia

Link	Data
https://www.javaworld.com/article/2077920/java-app-dev-rest-for-java-developers-part-1-it-s-about-the-information-stupid.html	05.09.2019
https://www.codecademy.com/articles/what-is-rest	05.09.2019
https://spring.io/guides/gs/rest-service/	06.09.2019
https://www.baeldung.com/jpa-join-column	01.10.2019
https://www.getpostman.com/downloads/	03.10.2019
https://www.baeldung.com/exception-handling-for-rest-with-spring	03.10.2019
https://www.baeldung.com/spring-request-param	09.10.2019
https://stackoverflow.com/questions/19896870/why-is-my-spring-autowired-field-null	08.10.2019
https://stackoverflow.com/questions/23921117/disable-only-full-group	10.10.2019
https://stackoverflow.com/questions/44647630/validation-failed-for-query-for-method-jp	15.10.2019
https://stackoverflow.com/questions/3325387/infinite-recursion-with-jackson-json-and-hibernate-jpa-issue/18288939#18288939	17.10.2019
https://codippa.com/how-to-autowire-objects-in-non-spring-classes/	18.10.2019

Link	Data
https://www.devglan.com/spring-security/spring-boot-security-rest-basic-authenticati	05.11.2019
https://www.baeldung.com/spring-security-basic-authentication	05.11.2019
https://www.baeldung.com/spring-security-with-maven	05.11.2019
https://www.javacodemonk.com/spring-security-5-there-is-no-passwordencoder-mapped-for-the-id-b0503f3d	07.11.2019
https://www.baeldung.com/spring-rest-template-multipart-uplo	22.11.2019
https://code-examples.net/en/q/24984	22.11.2019
https://docs.spring.io/spring/docs/current/javadoc-api/org.springframework.web.client.RestTemplate.html#exchange-java.lang.String-org.springframework.http.HttpMethod-org.springframework.http.HttpEntity-java.lang.Class-java.lang.Object...-	22.11.2019
https://tamasgyorfi.net/2015/03/27/posting-multipart-requests-with-resttemplate/	22.11.2019

9 Allegati

- Diari
- Gantt consuntivo
- CD con tutto il progetto e codice sorgente

Diari

Documentazione