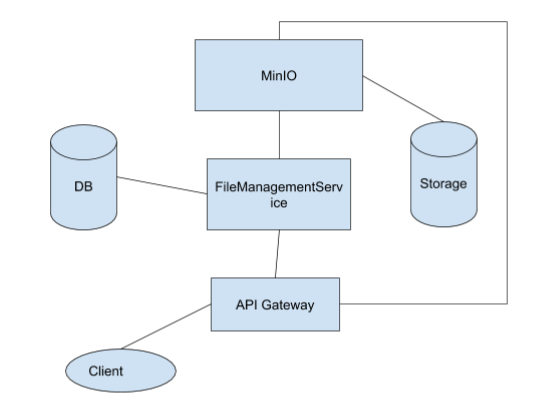
Object storage adapter

(relazione non ancora ultimata)

Coppolino Antonino O55000421

Labruna Mauro O55000429

- Homework 1 Object storage adapter



I componenti di nostro interesse nel nostro caso sono:

* DB mysql
* FileManagementService
* API Gateway
* MinIO

Per simulare l’interazione con un eventuale client abbiamo usato l’estensione di firefox RESTED.

**Creazione dei componenti e dockerizzazione**

Per prima cosa ci siamo occupati di creare lo scheletro dei componenti di nostro interesse ( project spring boot), dopo di che è stato fatto il package dei componenti stessi seguendo la corretta procedura con maven (clean-compile-package).

Completati i vari package si è proceduto con la scrittura dei vari dockerfile.

Il passo successivo è stato quello di scrivere il file docker-compose.yml in modo tale da containerizzare la nostra applicazione.

Una volta completata la scrittura del file docker-compose.yaml abbiamo eseguito il comando “docker-compose up”, quindi, in particolare abbiamo 4 container su cui girano i nostri 4 componenti.

**Implementazione features**

Una volta conteinerizzata l’applicazione ci siamo occupati di sviluppare le varie features.

Ogni richiesta http sarà fatta all’indirizzo dell’API Gateway, il quale si occuperà di smistare le richieste verso i componenti e fornire le risposte al client.

*POST /register*

Questa feature permette di registrare un nuovo utente, quindi, non necessita di alcuna forma di sicurezza.

Come detto precedentemente la richiesta iniziale sarà indirizzata all’API Gateway, il quale grazie alla sua configurazione la inoltrerà a FileManagementService.

(riportiamo il codice solo per questa prima feature)

.

route(p -> p.path("/register/\*\*")

.filters(f -> f.rewritePath("/register", "/user/register")  
 .filter(httpRequestInfo))  
 .uri(uriConfiguration.getUrl()))

L’implementazione di questa feature è stata fatta mediante il metodo register di UserService del progetto homework\_DSB

*POST /*

In questo caso è necessaria l’autenticazione, tutte le forme di sicurezza sono implementate nella classe AdapterConfiguration del progetto homework\_DSB.

.antMatchers(HttpMethod.*POST*,"/record/put").hasAnyAuthority("USER,ADMIN")

Se l’utente è gia registrato e l’autenticazione va a buon fine verrà inserito un nuovo record nel DB che conterrà il riferimento all’utente ed il filename.

Questa feature è stata implementata mediante il metodo saveRecord di RecordService del progetto Homework\_DSB

Poichè non abbiamo un client vero e proprio il file viene preso da una cartella locale interna al progetto h*omework\_DSB/*files

POST /id

Anche in questo caso è necessaria l’autenticazione, inoltre, l’id deve essere associato ad un record già inserito nel DB, altrimenti la nostra applicazione tornerà un messaggio di errore 400.

Se l’id è presente nel DB e il filename è valido, il file sarà caricato su MinIO dentro il bucket dell’utente autenticato, se il file non esiste l’applicazione risponderà con un messaggio di errore.

Per come abbiamo configurato MinIO, ogni utente ha il suo bucket privato, il nome del bucket sarà uguale al nickname dell’utente che deve essere @Unique.

L’implementazione di questa feature è stata fatta mediante il metodo checkRecord di RecordService del progetto Homework\_DSB.

GET /id

Anche in questo caso è necessaria l’autenticazione, dobbiamo però distinguere due casi:

* utente autenticato ha ruolo USER: l’utente può accedere solo ai record (e quindi ai file) di cui e proprietario, nel caso in cui l’utente provasse ad accedere al file di un altro utente l’applicazione risponderebbe con un messaggio di errore.

Se il processo invece andasse a buon fine, all’utente sarebbe ritornato il link mediante il quale può scaricare il file da MinIO

* Utente autenticato ha ruolo ADMIN: l’utente può accedere a qualsiasi file presente su MinIO.

L’implementazione di questa feature è stata fatta mediante il metodo getRecord() di RecordService del progetto homework\_DSB

GET /

Autenticazione necessaria, anche in questo caso dobbiamo fare la distinzione tra USER e ADMIN:

* utente con ruolo USER: l’applicazione tornerà la lista dei file di proprietà dell’utente
* utente con ruolo ADMIN: l’applicazione tornerà la lista di tutti i file presenti nel sistema.

L’implementazione di questa feature è stata fatta mediante il metodo getFileByUserRole() presente in MinioService nel progetto homework\_DSB

DELETE /id

Autenticazione necessaria, due distinzioni:

* utente ruolo ADMIN: può cancellare qualsiasi file
* utente ruolo USER: può cancellare solo i file di sua proprietà

Questa feature è stata implementata mediante il metodo deleteByUserRole presente in MinioService nel progetto homework\_DSB.

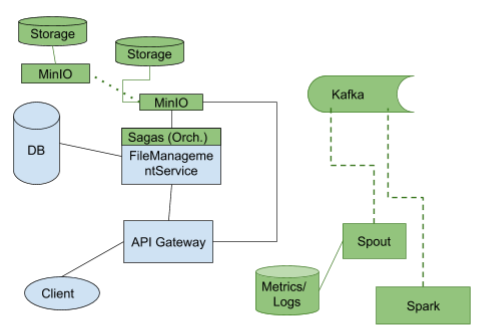
**Statistiche**

Per collezionare le statistiche delle varie richeste Http si è fatto uso di micrometer. La dipendenza è stata aggiunta al file pom.xml del modulo API Gateway. Per soddisfare le statistiche richieste nella consegna è stato implementato un custom filter in API Gateway mediante la classe HttpRequestInfo. In tale filtro viene istanziato un micrometer Timer, necessario per misurare il tempo di risposta alla richiesta http e ad esso sono stati aggiunti i tag: **router\_uri**, **http\_method**, **response**, **outcome** i quali indicano rispettivamente l’url a cui viene reindirizzata la richiesta, il metodo http utilizzato, la risposta del server ed il risultato della risposta.

Il custom filter è stato poi aggiunto ad ogni route del Bean **myRoutes** nella classe ApiGatewayApplication.

Per monitorare le statistiche si è deciso di utilizzare il sistema di monitoraggio Prometheus poiché supportato da micrometer e facilmente integrabile con con Spring. Prometheus permette inoltre di fare delle query sulle statistiche collezionate per ottenere nuove statistiche personalizzate (es. il numero di richieste al secondo).

**Homework 2**

****

Dopo aver installato minikube e kubectl ci siamo occupati di fare il porting su kubernetes.

Per eseguire il porting abbiamo installato kompose ed eseguito il comando kompose convert per poi fare l’apply in modo da poter deployare i nostri componenti.

In particolare per gestire le repliche abbiamo utilizzato il tipo StatefulSet.

Rispetto alla prima parte è stato introdotto un nuovo campo “status” nel record.

Per gestire le repliche abbiamo utilizzato la libreria InetAddress nel progetto homework\_DSB in MinioService, in modo da poter avere gli indirizzi delle varie repliche utili per inviare ed eliminare i file.

Il numero di repliche può essere facilmente cambiato dal file minio-deployment.yaml, l’applicazione si adatterà autonomamente alle nuove modifiche.