UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA:Tehnologia informației

An universitar 2019-2020

Proiect final de diploma licenta

Nume și prenume : Galan Ionut Andrei 1410A

Cuprins

Capitolul 1. Scopul documentului

Capitolul 2. Serviciu de gestionare a comentariilor

2.1. Implementare (DONE)

Serviciul de gestionare a comentariilor este o functionalitate importanta într-o aplicație de redare a continut media, așadar se pune un accent mare pe modul de implementare a comunicarii dintre clienți prin intermediul unui server de monitorizare și un stocare a comentariilor.

În aplicatiile monolit cea mai uzuala metoda de actualizare a noilor comentarii adaugate de către utilizatori este metoda de polling. Aceasta presupune trimiterea unor cerere permanente a clientului web către server pentru a verifica dacă au fost adaugate noi date în baza de date asociata. Metoda de polling mai este cunoscută și sub denumirea de pull technology care este asemănătoare cu push technology utilizata în transmisia unui mesaj clientului de catre server. Un exemplu este serviciul de mail care transmite notificari clientilor în momentul sosirii unui mesaj.

Metoda de polling se realizeaza in doua etape:

- controllerul ce va fi apelat repetat și răspunsul dat de acesta(funcție implementata la server)
- functia java script ce va realiza periodic o cerere către controller și care va face modificarile necesare în interfata.

O alta modalitate de implementare a acestei functionalitati este prin crearea unei comunicatii client/server prin intermediul protocolului WebSocket. Modul de implementare este crearea unei conexiuni websocket în spring boot prin protocolul STOMP[1].

WebSocket precum și HTTP sunt doua protocoale de comunicare bidirectionala, cu un canal full-duplex de comunicare intre un server și un client. Odată ce conexiunea intre client și server a fost realizata, are loc schimbul de date intre aceștia pana când una din instante închide conexiunea. Un avantaj important al utilizarii WebSocket-ului în pofida HTTP este faptul ca schimbul de informații intre cele doua entitati se realizeaza la frecventa mare și cu latenta mica deoarece în HTTP conexiunea este închisă după ce serverul a prelucrat cererea și a dat un răspuns. De asemenea, protocolul WebSocket este bidirecțional, adică un client se poate abona pentru un anumit eveniment, iar serverul poate publica evenimentul către client în funcție de disponibilitatea evenimentului din server. Server-Sent Events (SSE) este o tehnologie push server care permite unui browser să primească actualizări automate de la un server prin conexiune HTTP. Într-un eveniment trimis de server, o pagină web primește automat actualizări de la un server, iar serverul produce răspunsul într-un format text / eveniment-stream. Acesta functionalitate a fost introdusa în versiunea HTML 5.

[Referinta: https://www.devglan.com/spring-boot/spring-boot-websocket-example]

STOMP reprezintă un protocol bazat pe transmisia de text creat pentru a fi utilizat în arhitecturile MOM(message-oriented middleware). STOMP oferă un format interoperabil care permite clienților STOMP să discute cu orice broker de mesaje care acceptă protocolul. Spring framework oferă asistență implicită pentru acesta, dar se poate folosii orice protocol de mesagerie, cum ar fi RabbitMQ[2] sau ActiveMQ[3] care sunt implementari ale protocolului STOMP.

1 STOMP - Streaming Text Oriented Messaging Protocol.
2 RabbitMQ – Message Queue (protocol care are la baza o coada de mesaje)
3 ActiveMQ – Bazat pe JMS message(TextMessage or ByteMessage)

Utilizand Spring boot pentru configurarea WebSocketului trebuie aduagate următoarele dependinte:

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

2.2. Implementarea solutiei(DONE)

În cele ce urmează sunt prezentati pasii necesare pentru realizare conexiunii WebSocket într-o aplicație de tipul spring boot:

1) createa cai de conectare(*/replay*) a unui client de a se conecta la STOMP
2) configurarea STOMP
3) implementarea unui controller handler care va primi request-urile de la user și va trimire la toți useri înregistrați(subscribed)

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/replay").setAllowedOrigins("*");
    }

}
```

Metoda *configureMessageBroker* seteaza:

1. Creaza un message broker cu una sau mai multe destinatii pentru trimitearea și primirea mesajelor. În implementarea de mai sus exista o destinație mapata cu */topic*. În cazul în care erau create mai multe destinatii, acestea deservesc pentru mai multe tipuri de comentarii, private sau comentarii adresate de către un tip de user mai privilegiat. În aplicație se folosește o singura cale care deserveste toate comentariile trimise de client.
2. Difinirea rutei */app* este utilizata pentru a filtra datele de la adnotarea *@MessageMapping* care este implementata în controller. Controller-ul după ce proceseaza mesajul, acesta va fi trimis către broker.

3. Metoda *setAllowedOrigins(*)* este necesara deoarece WebSocketul accepta dar cererile venite de la aceeași origine. Cunoscut în literatura de specialitate sub denumirea de *same-origin policy* acesta este un mecanism de securitate ce nu permite comunicarea intre un client și server care au un domeniu diferit. Metoda respectiva dezactiveaza setarea implicita a implementarii WebSocket-ului și permite comunicarea intre domenii diferite ale celor doua instante de comunicare Mecanismul de securitate fiind implementata separat.

```
@Controller
public class CommentWebSocketController {

    @Autowired
    CommentService commentService;

    @MessageMapping("/comment")
    @SendTo("/topic/comment")
    public Comment addComment(Comment comment) throws Exception {

        commentService.save(comment);

        Thread.sleep(1000); // simulated delay
        return Comment.builder()
                .content(HtmlUtils.htmlEscape(comment.getContent()))
                .idMovie(comment.getIdMovie())
                .idUser(comment.getIdUser())
                .build();
    }
}
```

Clasa *CommentWebSocketController* va primi mesajul de la un utilizator, continutul mesajului va fi transferat sub forma de JSON și va fi mapat automat la un obiect *Comment.* Metoda *addComment* va face broadcast către ceilalți clienți înregistrați prin calea */topic/comment*. Comentariul adaugat de către un utilizator va fi salvat și în baza de date prin apelul metodei *commentService.save(comment).*

Diagrama completă de funcționare este descrisa în schema din figura de mai jos.
[https://www.toptal.com/java/stomp-spring-boot-websocket]

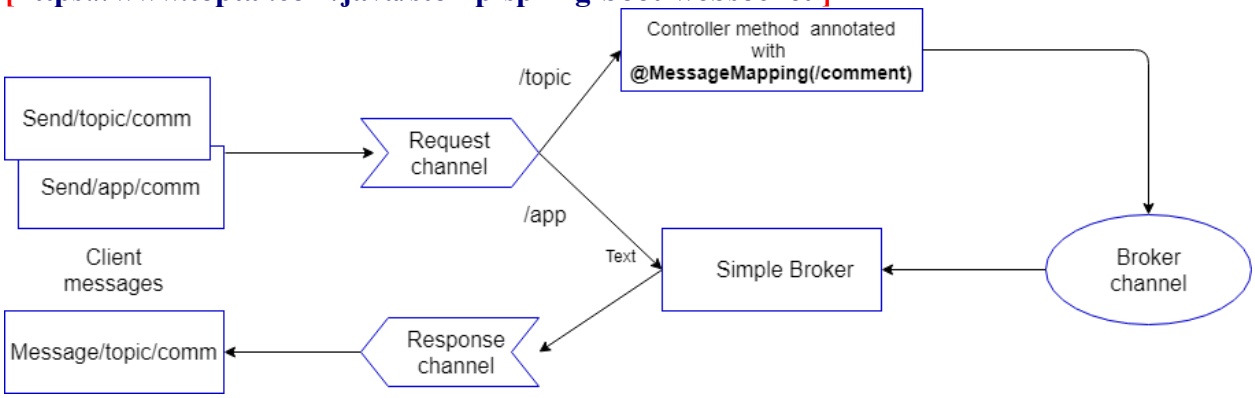

Figure 1: Transmiterea comentariilor catre server.

Capitolul 3. Serviciu de gestionare continut media

3.1. Prezentare generala (Progress..)

Gestionarea continutului media mereu a fost o problemă de interes și în care s-a investit constat pentru a se dezvolta protocoale de comunicatie cât mai performate și cât mai ușor de utilizat.

3.2. Prezentarea protocoalelor de video streaming(DONE)

3.2.1. Real-Time Messaging Protocol (RTMP)

RTMP a fost dezvoltat inițial de către Macromedia cu mult timp în urma iar în zilele noastre RTMP este utilizat pe scară largă. Cu toate acestea, astăzi este folosit mai ales pentru incarcarea fluxurilor live. În termeni simpli, atunci când configurați codificatorul pentru a vă trimite fluxul video pe platforma de găzduire video, acel videoclip va ajunge pe CDN[1] prin protocolul RTMP. Cu toate acestea, acel conținut ajunge în cele din urmă la vizualizatorul final într-un alt protocol - de obicei protocolul de streaming HLS. Astăzi, RTMP este depreciat în cea ce privește utilizare ca protocol de streaming video. Acest lucru se datorează faptului că depinde de pluginul Flash, care a fost afectat de probleme de securitate și nu mai este atât de folosit. Pluginul Flash este utilizat pentru redarea continutului media în pagini web cât și pe dispozitivele mobile. Adobe, cei care dețin pluginul de Flash Player începând din anul 2020 nu vor mai oferii suport pentru Flash. Locul acestuia fiind luat de HTML5 care încerca sa acopere toate functionalitatile plaginului Flash.

3.2.2. Real-Time Streaming Protocol (RTSP)

RTSP, poate un protocol de streaming video mai puțin cunoscut, Real-Time Streaming Protocol (RTSP) a fost publicat pentru prima dată în 1998. RTSP a fost dezvoltat pentru a controla serverele de streaming în diferite sistemele, precum cele de divertisment și în special cele de comunicații. În 2016, apare o noua versiune RTSP 2.0. În general, este cunoscut sub numele de protocol de streaming video pentru stabilirea și controlul sesiunilor media între puncte finale. În multe cazuri RTSP este similar protocolului HTL Live Streaming (HLS), pe care îl vom descrie mai jos. Cu toate acestea, transmiterea datelor nu este realizata utilizand doar acest protocol. Serverele RTSP funcționează deseori în combinație cu Protocolul de transport în timp real (RTP) și Protocolul de control în timp real (RTCP) pentru a furniza fluxuri media.

RTP oferă funcții de transport de rețea end-to-end adecvate pentru aplicații care transmit date în timp real, cum ar fi audio, video sau date de simulare, prin servicii de rețea multicast sau unicast. RTP nu garantează calitatea serviciilor pentru servicii în timp real. Transportul de date este utilizat împreuna cu un protocol de control (RTCP) pentru a permite monitorizarea furnizarii datelor într-un mod scalabil către rețele multicast mari și pentru a oferi funcționalități minime de control și identificare. RTP și RTCP sunt concepute pentru a fi independente de transportul de bază și straturile de rețea. [https://tools.ietf.org/html/rfc3550#section-1.1]

RTSP a fost conceput pentru a sprijini streaming-ul cu latență scăzută și poate fi o alegere bună pentru fluxurile de date mici, cum ar fi fluxurile de date preluate de la camere video (de

1 CDN – Content delivery network

exemplu, camere de securitate), dispozitive IoT (de exemplu, drone controlate de laptop) și SDK-uri mobile. Un neajuns semnificativ este faptul ca RTSP este limitat în ceea ce privește utilizarea și suportul în browser.

3.2.3. Dynamic Adaptive Streaming over HTTP (MPEG-DASH): up-and-coming protocol

MPEG-DASH este unul dintre cele mai noi protocoale. Deși nu este utilizat pe scară largă, acest protocol are câteva avantaje mari. În primul rând, acceptă streaming-bitrate adaptiv. Aceasta înseamnă că utilizatorii vor primi întotdeauna continutul video la cea mai bună calitate pe care o poate susține viteza lor actuală de conectare la internet. Aceasta poate fluctua din secunda în secundă, iar DASH poate ține pasul.

Protocolul de streaming MPEG-DASH rezolvă unele probleme persistente de mult timp legate de livrarea și compresia continutului. Un alt avantaj este că MPEG-DASH este „codec agnostic” - poate fi utilizat cu aproape orice format de codificare. De asemenea, acceptă extensiile media criptate (EME) și extensia sursă media (MSE), care sunt API-uri bazate pe standarde pentru gestionarea drepturilor digitale bazate pe browser (DRM).

Astazi, MPEG-DASH este utilizat pe scară largă doar de o anumita clasa de emițători, totuși acesta clasa fiind redusa în comparație cu clasa de emitatori care au la baza HLS. Cu toate acestea, va fi tehnologia standard în viitor. Deocamdată, exista o problema în ceea ce privește compatibilitățiile (de exemplu, dispozitivele Apple Safari și iOS nu o acceptă) și alte probleme.

3.2.4. Microsoft Smooth Streaming (MSS)

Microsoft Smooth Streaming (MSS) a fost introdus în 2008, MSS a fost parte integrantă a Jocurilor Olimpice de vară din acel an. Cu toate acestea, în zilele noastre nu este foarte utilizat, cu excepția dezvoltatorilor de la Microsoft și a celor care lucrează în ecosistemul Xbox.

Smooth Streaming acceptă streaming-bitrate adaptiv și include câteva instrumente solide pentru DRM. În general, este o metodă de livrare media hibridă care funcționează precum streamingul, dar se bazează pe descărcarea progresivă HTTP.

Managementul restricțiilor digitale(DRM) este practica de a impune restricții tehnologice prin care se controlează ceea ce utilizatorii pot face cu un conținut media. Când un program este conceput pentru a împiedica să fie copiat sau se doreste imposibilitatea partajarii unei melodii, citirea unui ebook de pe alt dispozitiv sau jocul cu un singur player fără conexiune la Internet, aceste restricții se fac de catre DRM. [https://www.defectivebydesign.org/what_is_drm_digital_restrictions_management]

Protocolul MSS este utilizat în crearea aplicatiilor specifice Windows cât și în cadrul platformei Xbox. Acesta nu este recomandat să fie utilizat ca și protocol principal de streaming video.

3.2.5. HTTP Dynamic Streaming (HDS)

HTTP Dynamic Streaming (HDS) a reprezentat intrarea Adobe în lumea protocolului de streaming, acesta fiind succesorul RTMP. Ca și RTMP, HDS este un protocol de streaming bazat pe Flash. Cu toate acestea, adaugă și suport pentru streaming adaptiv și are o reputație de înaltă calitate. HDS este, de asemenea, unul dintre protocoalele mai bune atunci când vine vorba de latență. Pe de altă parte, latența nu este la fel de scăzută ca în cazul RTMP, deoarece la acest protocol se adauga si un proces de fragmentare și criptare. Acesta îl fac mai puțin popular pentru streaming și alte evenimente în care contează timpul și viteza de redare.

În general, nu este recomandat utilizarea acestui protocul pentru crearea de servere de streaming din motiv ca utilizeaza Flash, acesta fiind o piedica în ceea ce privește viteza de redare și transmitere a datelor media.

3.2.6. HTTP Live Streaming (HLS)

HTTP Live Streaming sau HLS a fost lansat inițial de Apple în 2009 pentru a le permite să renunțe la tehnologia Flash de pe iPhone. De atunci, și mai ales acum, HLS a devenit cel mai utilizat protocol de streaming.

Este cel mai utilizat protocol deoarece browserele, televizoarele inteligente și dispozitivele mobile Android și iOS toate acceptă HLS. Player-ul *video* HTML5 suportă în mod nativ HLS, în comparație cu HDS și RTMP. Aceasta permite unui flux să ajungă la un număr cât mai mare de spectatori, ceea ce face din HLS cel mai sigur protocol de astăzi pentru scalarea unui flux live către publicul larg.

În ceea ce privește caracteristicile, standardul HLS acceptă, de asemenea, streamingul de biți adaptivi, oferind dinamic cea mai bună calitate video posibilă în orice moment. Comform ultimelor actualizari, acest standard acceptă cel mai recent și cel mai mare codec, H.265[1], care oferă de două ori calitatea videoului la aceeași dimensiune de fișier ca H.264.

În prezent, singurul dezavantaj al HLS este că latența poate fi relativ mare. Cu toate acestea, există metode pentru reducerea latenței HLS.[https://www.dacast.com/blog/video-streaming-protocol/]

3.3. HLS Streaming Protocol

Notiuni teoretice despre HLS.
[https://www.cloudflare.com/learning/video/what-is-http-live-streaming/]
[https://dzone.com/articles/hls-streaming-protocol]

3.4. Implementare

Serviciul de gestionare continut media create are la baza protocolul HLS și este realizat prin intermediul unui cluster de microservicii stocate pe servere diferite. Microserviciile au rolul de a distribui sarcinile în funcție de tipul cererilor venite de la utilizator. Așadar se identifica urmatoarele microservicii:

1. Serviciul de spargere în bucăți a continutului media încărcat
2. Serviciul de scriere în baza de date NoSQL a bucatilor de continut
3. Serviciul de scriere în baza de date SQL a metadatelor continutului
4. Serviciul de încărcare a continutului selectat de utilizator

Diagrama de componente din Figura 2 prezinta modelul după care se realizeaza comunicarea intre microserviciul de împărțire în bucăți, și cele doua microservicii de stocare în bazele de date aferente. În acesta diagrama mai apare un serviciu sub numele de Încărcare din browser a locatiei continutului, evidentiind faptul ca spargerea în bucăți nu se face în browser prin intermediul limbajului aferent ci prin intermediul serviciul de împărțire în bucăți care va fi instalat și va rula pe mașina utilizatorului. În cele din urma utilizatorul va transmite din browser doar calea absolută a fisierului.

1 H.265 – codec video, versionea imbunatatita a H.264

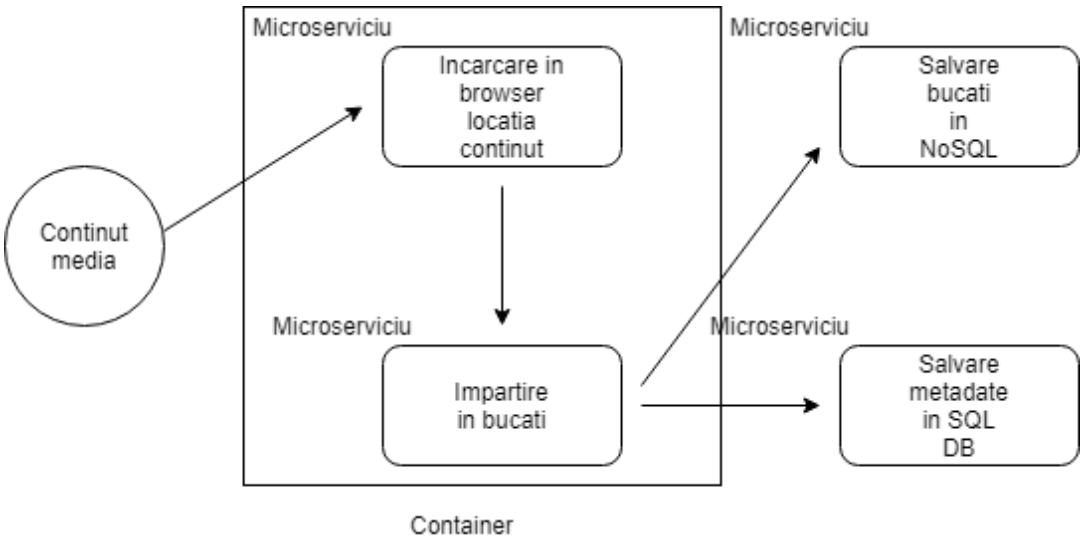

Figure 2: Diagrama de componente pentru serviciul de spargere in bucati.

3.4.1. Serviciul de spargere în bucăți a continutului media încărcat

Ideea implementarii acestui serviciu pornește de la premisa ca utilizatorul deține un continutul media care ulterior dorește să fie încărcat și distribuit celorlalți utilizatori conectați pe platforma. În prima faza continutul este stocat static pe dispozitivul utilizatorului. Acesta prin intermediul serviciului oferit încarca continutul pe platforma.

Cum se încarca continutul ?

O prima analiza a acestei probleme de încărcare a unui continut destul de consistent aduce în atenția programatorului doua moduri de încărcare:

1. Încărcarea fisierului la dimensiunea lui normala. În cazul de fata timpul de încărcare a fisierului nu ar avea vreo influența majora asupra performantei aplicației deoarece scopul este ca acel fisier să fie încărcat pe un server care ulterior sa răspundă utilizatorilor cu acel continut cerut. Problema în aceasta abordare apare în momentul incarcarii continutului media. Datorită faptului ca acest continut este de natura media utilizatorul se aștepta ca acesta să fie disponibil din momentul în care acesta acceseaza linkul de redare a continutui. Încărcarea fisierului la dimensiunea lui reala nu va fi o soluție buna deoarece utilizatorul trebuie sa aștepta descarcarea lui integrala apoi redarea acestuia. Rezulta o performanța scazuta chiar și dacă fișierul media va fi compresat și transferat.
2. Încărcarea fisierului și stocarea acestuia în bucăți de dimensiune fixa este o alta soluție mult mai performanta în momentul redarii continutului video. Practic în timpul redarii continutului video are loc și încărcarea acestuia. Principiul de spargere în bucăți mai mici sta și la baza HLS.

În prima faza continutul fisierului este de un anumit format *webm* ulterior existand posibilitatea creerii unui serviciu care sa permita încărcarea fisierelor de mai multe tipuri. Alegerea acestui format nu a permis redarea continului media în tag-ul video aferent din HTML5.

Concluzii

Bibliografie

Anexe.

Anexa 1.