



UNIVERSITATEA DIN  
BUCUREȘTI

FACULTATEA DE  
MATEMATICĂ ȘI  
INFORMATICĂ



SPECIALIZAREA INFORMATICĂ

Lucrare de licență

APLICAȚIE WEB PENTRU VIDEOCLIP-URI CU  
FUNCȚII DE ÎNȚELEGEREA VORBIRII ȘI  
REGĂSIRE PE BAZĂ DE TEXT

Absolvent

Trifan Robert-Gabriel

Coordonator științific

Prof. Dr. Ionescu Radu

București, iunie 2024

## **Rezumat**

Popularitatea videoclipurilor a avut o creștere constantă în ultimii ani, reprezentând 82.5% din traficul web în 2023. Statistici recente arată că oamenii petrec în media 17 ore pe săptămâna vizionând videoclipuri online, acestea fiind cu 52% mai predispuse să fie distribuite pe rețelele de socializare decât alte tipuri de conținut. [3]

În acest context, lucrarea de față își propune să contribuie la creșterea accesibilității și personalizării conținutului video, prin dezvoltarea unei aplicații web care permite adăugarea de subtitrări, căutarea videoclipurilor pe bază de text și clasificarea acestora în funcție de conținutul lor. Aplicația oferă utilizatorilor posibilitatea de a vizualiza subtitrările în timp real și de a căuta cuvinte cheie în metadatele videoclipurilor precum titlu, descriere, topic și subtitrare.

## **Abstract**

The popularity of videos has been steadily increasing in recent years, representing 82.5% of web traffic in 2023. Recent statistics show that people spend an average of 17 hours a week watching online videos, which are 52% more likely to be shared on social networks than other types of content. [3]

In this context, this work aims to contribute to increasing the accessibility and personalization of video content, by developing a web application that allows the addition of subtitles, searching for videos based on text and classifying them according to their content. The application offers users the possibility to view subtitles in real time and to search for keywords in the metadata of videos such as title, description, topic and subtitle.

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>5</b>
1.1	Motivație . . . . .	5
1.2	Domenii abordate . . . . .	5
1.3	Structura lucrării . . . . .	6
<b>2</b>	<b>Concepte teoretice despre învățare automată</b>	<b>7</b>
2.1	Recunoașterea vorbirii . . . . .	7
2.1.1	Semnalul audio . . . . .	7
2.1.2	Arhitectura modelului . . . . .	9
2.1.3	Setul de date . . . . .	14
2.1.4	Antrenarea modelului . . . . .	15
2.1.5	Îmbunătățire cu un model de limbaj . . . . .	15
2.2	Clasificarea videoclipurilor . . . . .	17
2.2.1	Arhitectura modelului . . . . .	18
2.2.2	Setul de date . . . . .	19
2.2.3	Antrenarea modelului . . . . .	19
2.2.4	Îmbunătățirea clasificării . . . . .	21
2.3	Concluzie . . . . .	21
<b>3</b>	<b>Prezentarea aplicației</b>	<b>22</b>
3.1	Arhitectura aplicației . . . . .	22
3.2	Frontend . . . . .	23
3.2.1	React . . . . .	23
3.2.2	Autentificare . . . . .	24
3.2.3	Profil . . . . .	24
3.2.4	Video . . . . .	25
3.2.5	Căutare . . . . .	30
3.2.6	Conversație . . . . .	30
3.2.7	Temă . . . . .	32
3.2.8	Rute protejate . . . . .	32
3.3	Backend . . . . .	32

3.3.1	Express Server - Procesarea cererilor . . . . .	32
3.3.2	Socket.io Server - Comunicare în timp real . . . . .	40
3.3.3	Flask Server - Recunoașterea vorbirii . . . . .	41
3.3.4	Flask Server - Clasificarea videoclipurilor . . . . .	45
3.3.5	Flask Server - Întrebări și răspunsuri pentru videoclipuri . . . . .	46
3.4	Baze de date . . . . .	46
3.4.1	MongoDB . . . . .	46
3.4.2	Elasticsearch . . . . .	48
3.5	Servicii . . . . .	49
3.5.1	Cron Jobs . . . . .	49
3.6	Încărcare . . . . .	51
3.6.1	Docker . . . . .	51
3.6.2	Dockerizarea aplicației . . . . .	53
<b>4</b>	<b>Concluzie</b>	<b>54</b>
4.1	Concluzie . . . . .	54
4.2	Perspective . . . . .	55
	<b>Bibliografie</b>	<b>57</b>

# Capitolul 1

## Introducere

### 1.1 Motivație

Motivul pentru care am ales această temă îl reprezintă nevoia de subtitrări pentru videoclipuri, în special pentru filme, dar și pentru tutoriale sau alte tipuri de conținut video, a căror înțelegere este îngreunată de calitatea slabă a sunetului sau de faptul că sunt într-o limbă străină. Astfel, în această lucrare, am ales să abordez problema subtitrărilor prin intermediul unui model de recunoaștere a vorbirii, care extrage audio dintr-un videoclip și generează textul corespunzător.

De asemenea, am ales să abordez și problema căutării videoclipurilor, care constă în căutarea cuvintelor cheie în metadate precum titlu, descriere, topicuri sau chiar în conținutul videoclipului, folosind o bază de date special concepută pentru acest scop, Elasticsearch. [5]

Ideile menționate mai sus vor fi integrate într-o aplicație web folosind ansamblul de tehnologii MERN (MongoDB, Express.js, React.js, Node.js), scrise în TypeScript, împreună cu 3 servere de Flask în Python pentru recunoașterea vorbirii, clasificarea videoclipurilor și răspunderea la întrebări legate de conținutul videoclipurilor.

### 1.2 Domenii abordate

Această lucrare abordează 5 domenii principale:

- **Procesarea semnalelor audio** - pentru extragerea audio din videoclipuri și recunoașterea vorbirii
- **Procesarea limbajului natural** - pentru clasificarea videoclipurilor în funcție de conținutul lor
- **Frontend** (*React.js*) pentru interfața cu utilizatorul și pentru a oferi acces la funcționalitățile sistemului

- **Backend** - pentru gestionarea cererilor prin intermediul unui API, cu ajutorul a 3 servicii principale:
  - **Server** (*Node.js, Express.js*) pentru gestionarea cererilor și a răspunsurilor
  - **Recunoașterea vorbirii** (*Flask*) pentru gestionarea cererilor de recunoaștere a vorbirii
  - **Clasificarea videoclipurilor** (*Flask*) pentru clasificarea videoclipurilor în funcție de conținutul lor
  - **Întrebări și răspunsuri** (*Flask*) folosește API-ul de la ChatGPT pentru a răspunde la întrebări legate de conținutul videoclipurilor
- **Baze de date**
  - **MongoDB** pentru stocarea metadatelor videoclipurilor
  - **Elasticsearch** pentru căutarea videoclipurilor în funcție de cuvintele cheie

## 1.3 Structura lucrării

Vom împărți această lucrare în 2 capitole principale:

- **Concepte teoretice despre învățare automată** - în care voi aborda *recunoașterea vorbirii* și *clasificarea videoclipurilor* și voi prezenta setul de date folosit, arhitectura modelului, antrenarea și evaluarea acestuia, precum și procesările ulterioare.
- **Prezentarea aplicației** - în care voi aborda partea de frontend, backend și baze de date, precum și detaliile tehnice ale implementării.

# Capitolul 2

## Concepte teoretice despre învățare automată

### 2.1 Recunoașterea vorbirii

Pentru a putea recunoaște vorbirea dintr-un videoclip, am ales să folosesc arhitectura *wav2vec 2.0* [2] dezvoltată de Facebook AI Research. Am folosit atât modelul *facebook/wav2vec2-base-960h* antrenat pe setul de date *LibriSpeech* [10], cât și modelul preantrenat *facebook/wav2vec2-base* pe care le-am antrenat pe seturile de date *Mini LibriSpeech* (subset din LibriSpeech) și *Common Voice Delta Segment 16.1* (subset din Common Voice) [1].

#### 2.1.1 Semnalul audio

Înainte de a intra în detalii despre modelele folosite, este important să înțelegem în ce constă semnalul audio, cum este reprezentat în memorie și cum este procesat.

Definim un convertor analog-digital ca fiind un dispozitiv care transformă semnalul recepționat de la microfon (variații în presiunea aerului/unde sonore) într-un semnal digital. Practic, convertește semnalul analogic continuu într-un semnal digital discret care poate fi stocat și procesat de un calculator. Acest proces implică conceptul de cuantizare și introduce o eroare de cuantizare care depinde de rezoluția convertorului. În acest sens, convertorul analog-digital convertește periodic semnalul analogic, eșantionând semnalul la intervale de timp egale.

Perioada de timp dintre două eșantioane consecutive este cunoscută sub numele de **frecvență de eșantionare** și este măsurată în Hz. Cu cât frecvența de eșantionare este mai mare, cu atât semnalul digital este mai apropiat de semnalul analogic original. Această frecvență măsoară numărul de sample-uri pe secundă.

Conceptul de **rezoluție** indică numărul de valori discrete distincte pe care le poate reprezenta un convertor analog-digital. Cu cât rezoluția este mai mare, cu atât semnalul digital este mai apropiat de semnalul analogic original.

De exemplu, un convertor analog-digital cu o rezoluție de 8 biți poate encoda un semnal analogic în  $2^8 = 256$  de valori discrete. Figura de mai jos ilustrează procesul de conversie analog-digital și digital-analog. (Fig. 2.1)

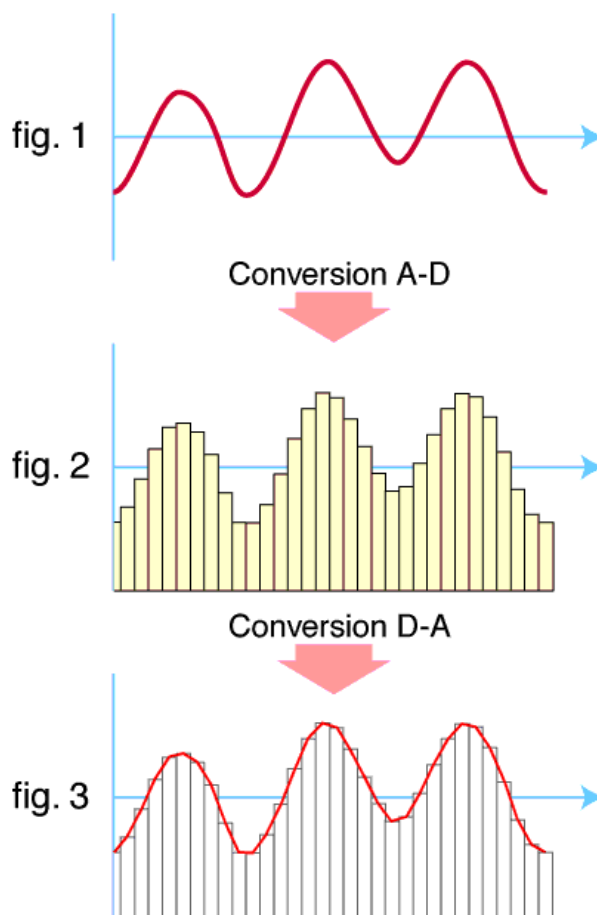


Figura 2.1: Procesul de conversie analog-digital și digital-analog<sup>1</sup>

Acum că am definit cum sunt interpretate semnalele audio, putem să trecem la procesarea acestora. Definim ce semnifică valorile obținute (Fig. 2.2):

- **Amplitudinea:** reprezintă presiunea undelor sonore măsurată la un moment de timp. Cu cât amplitudinea este mai mare, cu atât sunetul este mai puternic.
- **Amplitudinea zero:** valoarea 0 reprezintă linia de bază a semnalului audio, punctul principal de referință după care se măsoară amplitudinea.
- **Valori pozitive și negative:** valorile sunt reprezentate în formatul PCM (*Pulse Code Modulation*), unde valorile pozitive sunt reprezentate de valori între 0 și 1, iar valorile negative sunt reprezentate de valori între 0 și -1.

<sup>1</sup>Imagine preluată de pe Wikipedia la adresa: [https://en.wikipedia.org/wiki/Analog-to-digital\\_converter](https://en.wikipedia.org/wiki/Analog-to-digital_converter)



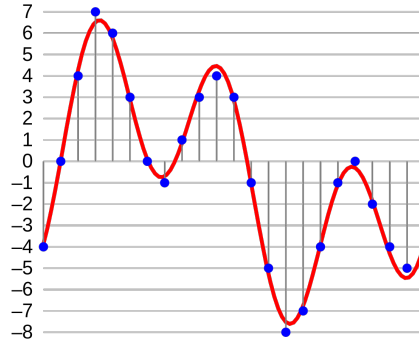


Figura 2.2: Reprezentarea semnalului audio în format PCM<sup>1</sup>

De aici încolo vom lucra cu semnalul digital în Python folosind librăriile *soundfile*, *librosa* și *numpy* pentru a procesa secvențele audio.

### 2.1.2 Arhitectura modelului

Modelul *wav2vec 2.0* este un model de învățare profundă alcătuit din 4 componente principale: **Codificator de Caracteristici Latente** (*Latent Feature Encoder*), reprezentat de o *Rețea Neuronală Convoluțională*, **Rețea pentru Context** (*Context Network*), constă în partea de Encodare a Transformer-ului, **Modulul de Cuantizare** (*Quantization Module*), aplică funcția *Gumbel Softmax* și **Pierdere Contrastivă**. (Fig. 2.3)

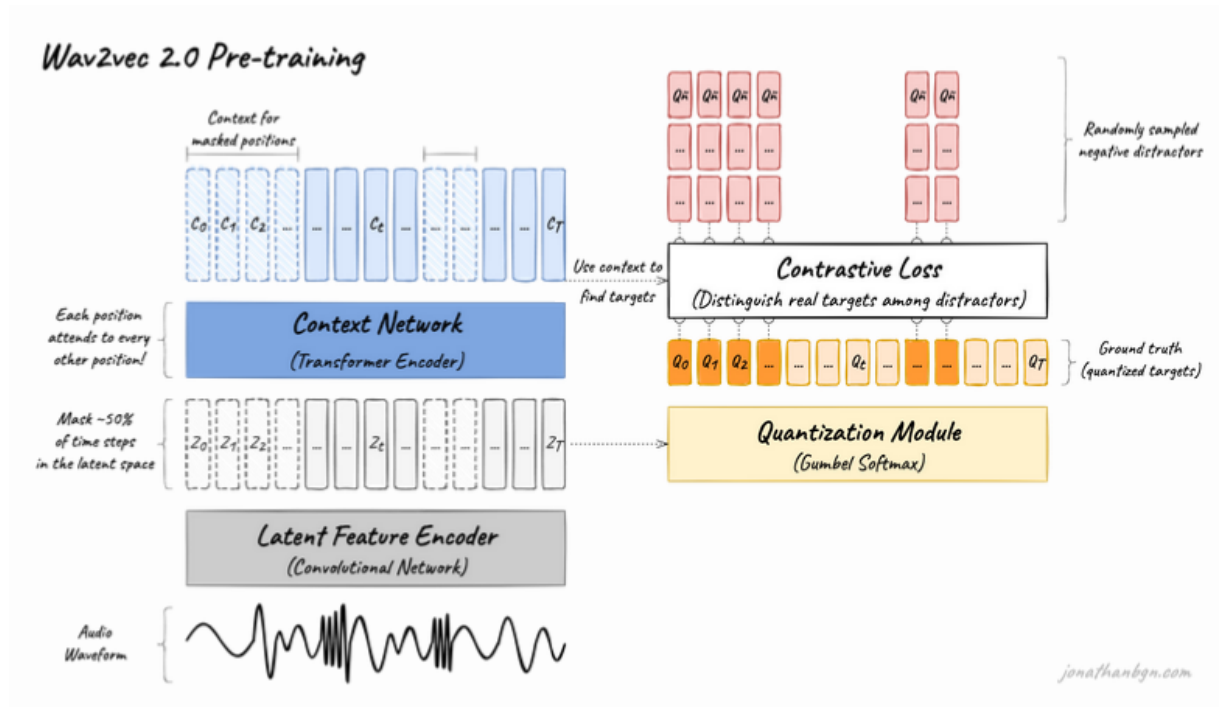


Figura 2.3: Arhitectura modelului *wav2vec 2.0* <sup>2</sup>

<sup>1</sup>Imagine preluată de pe Wikipedia la adresa: [https://en.wikipedia.org/wiki/Audio\\_bit\\_depth](https://en.wikipedia.org/wiki/Audio_bit_depth)

<sup>2</sup>Imagine preluată de pe site-ul lui Jonathan Bgn, "Illustrated Wav2Vec 2.0", disponibil la: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>.

## Codificator de Caracteristici Latente

**Codificatorul de caracteristici latente** (*Latent Feature Encoder*) este prima parte a modelului *wav2vec 2.0* și se ocupă cu prelucrarea semnalului audio (explicat anterior în secțiunea 2.1.1). Astfel, Codificatorul de Caracteristici Latente procesează semnalul audio din forma lui inițială folosind o normalizare de medie 0 și deviație standard 1. Apoi, continuă cu 7 blocuri convoluționale, fiecare alcătuit din convoluții 1-dimensionale, normalizare la nivel de strat și activări GELU (*Gaussian Error Linear Unit*).

Dimensiunea canalelor, a filtrului și a lungimii pasului sunt prezentate în figura de mai jos (Fig. 2.4). În această etapă, modelul încearcă să înțeleagă legătura dintre valorile apropiate ale semnalului audio și să extragă caracteristici latente care vor fi folosite ulterior. Arhitectura blocurilor convoluționale este structurată astfel:

- **Blocul 1:** dimensiunea filtrului 10, lungimea pasului 5, canale 512
- **Blocurile 2-5:** dimensiunea filtrului 3, lungimea pasului 2, canale 512
- **Blocurile 6-7:** dimensiunea filtrului 2, lungimea pasului 2, canale 512

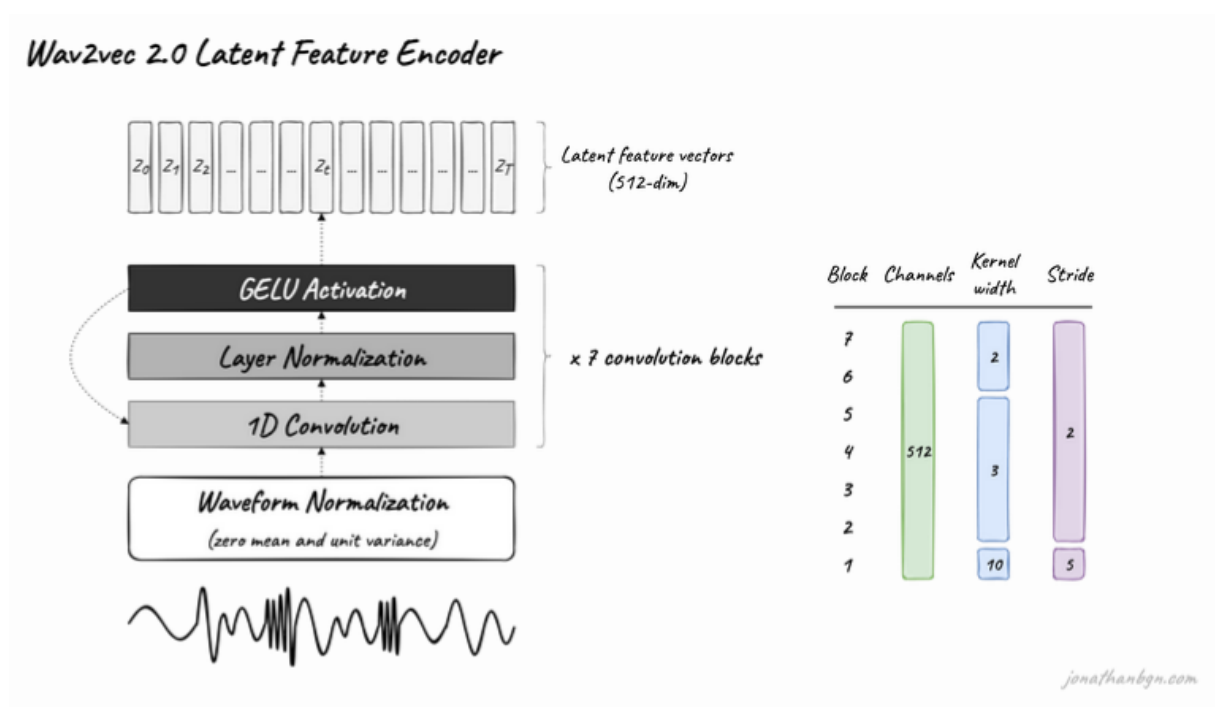


Figura 2.4: Arhitectura componentei *Latent Feature Encoder*<sup>1</sup>

Se observă cum primul bloc convoluțional are dimensiunea filtrului 10 și lungimea pasului 5, fiind cele mai mari valori din cele 7 blocuri. Intuitiv, acest bloc încearcă să acopere o porțiune mai mare din semnalul audio, în timp ce blocurile următoare (cu dimensiuni mai mici) încearcă să înțeleagă detalii mai fine din semnalul audio. În

<sup>1</sup>Imagine preluată de pe site-ul lui Jonathan Bgn, “Illustrated Wav2Vec 2.0”, disponibil la: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>.

## Rețea pentru Context

Componenta **Rețea pentru context** (*Context Network*) reprezintă inovația adusă de modelul *wav2vec 2.0* față de predecesorul său *wav2vec* deoarece folosește un codificator de tip *Transformer* care oferă o reprezentare contextuală mult mai bună decât codificatorul convoluțional folosit anterior.

Asfel, caracteristicile extrase de componenta **Codificator de Caracteristici Latente** sunt proiectate într-un spațiu latent de dimensiune 768 pentru modelul *BASE*, respectiv 1024 pentru modelul *LARGE* pentru a putea fi procesate de rețeaua de tip *Transformer*.

Un alt aspect important, înainte de a trece prin rețeaua de tip *Transformer*, îl reprezintă *encodarea pozițională* a caracteristicilor latente. Această encodare pozițională adaugă informații despre poziția relativă a caracteristicilor la nivel de secvență, corelând astfel caracteristicile cu poziția lor în secvența audio.

În final, caracteristicile sunt trecute prin codificatorul de tip *Transformer* care, prin *mecanismul de atenție*, învață legături între caracteristici și poziția lor în secvență, oferind o reprezentare contextuală cât mai bună a semnalului audio.

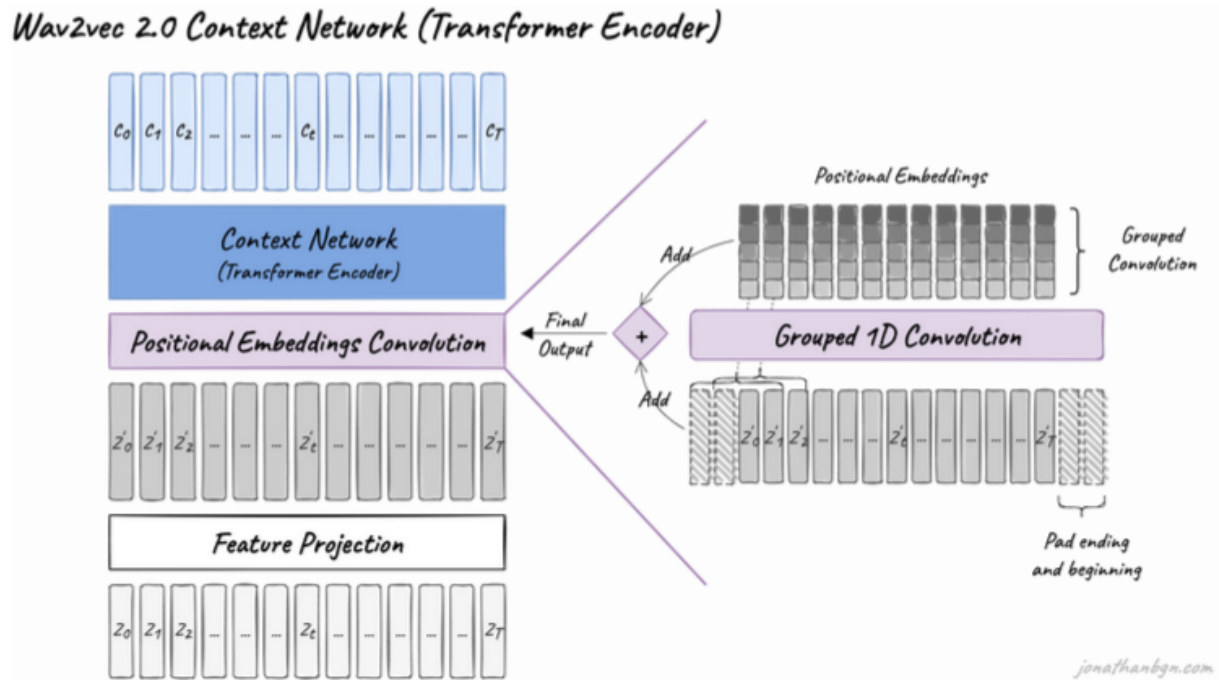


Figura 2.5: Arhitectura componentei *Context Network*<sup>1</sup>

<sup>1</sup>Imagine preluată de pe site-ul lui Jonathan Bgn, “Illustrated Wav2Vec 2.0”, disponibil la: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>.

## Modulul de cuantizare

Din cauza naturii continue a semnalului audio, modelul *wav2vec 2.0* nu poate folosi în mod direct un vocabular discret de simboluri, așa cum întâlnim în limbajul scris. Astfel, autorii modelului propun conceptul de **codebook** alcătuit **codewords** pentru a discretiza semnalul audio. Intuitiv, ne putem gândi la *codebook* ca la un vocabular de sunete fonetice reprezentative pentru semnalul audio. Deoarece un semnalul audio poate conține mai multe sunete fonetice, autorii propun folosirea a  $G$  *codebooks* fiecare cu câte  $V$  *codewords* creând astfel o matrice denumită *matricea de cuantizare*.

După înmulțirea caracteristicilor din spațiul latent cu matricea de cuantizare, se aplică funcția **Gumbel Softmax** peste logits și se obțin astfel vectori care conțin valoarea 1 pe poziția corespunzătoare sunetului fonetic și 0 pe restul pozițiilor. Spre deosebire de funcția *Softmax*, funcția *Gumbel Softmax* adaugă zgomot de medie 0 și deviație standard 1, zgomot reglat cu ajutorul temperaturii, care ajută modelul să exploreze mai multe sunete fonetice în timpul antrenării. De asemenea, funcția *Gumbel Softmax* este o funcție diferențiabilă care permite antrenarea modelului prin tehnica de **coborârii pe gradient**.

Vectorii menționați anterior indică poziția sunetului fonetic în *codebook* și sunt înmulțiți apoi cu o matrice de proiecție pentru a redimensionarea caracteristicile la dimensiunea ieșirilor codificatorului de tip Transformer.

### Wav2vec 2.0 Quantization Module

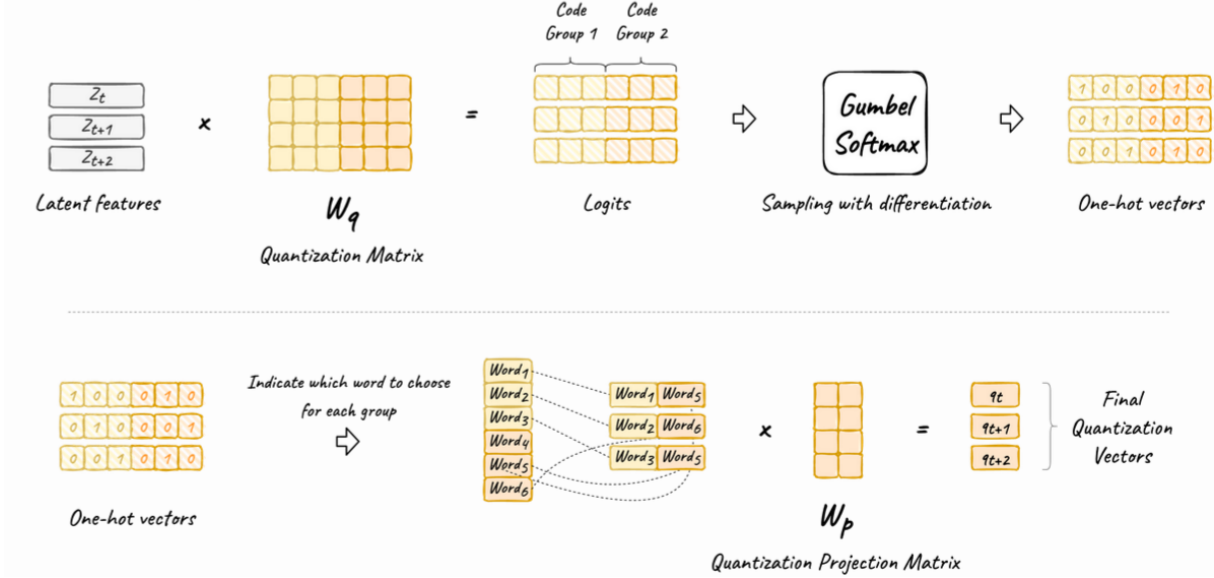


Figura 2.6: Arhitectura componentei *Quantization Module* <sup>1</sup>

<sup>1</sup>Imaginile au fost preluate de pe site-ul lui Jonathan Bgn, "Illustrated Wav2Vec 2.0", disponibil la: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>.

## Pierdere contrastivă

În special pentru partea de preantrenare a modelului, în care scopul principal este înțelegerea dependențelor între caracteristicile latente ale semnalului audio, autorii propun folosirea unei funcții de pierdere contrastivă.

Astfel, pentru a forța modelul să învețe reprezentări semantice, se folosește o mască care ascunde  $\sim 50\%$  din vectorii proiectați din spațiul latent înainte să fie trecuți prin **Rețeaua pentru Context**. În procesul de învățare, vectorii mascați vor fi înlocuiți apoi cu reprezentările învățate de model.

Pentru fiecare poziție mascată, se aleg uniform aleator 100 de exemple negative de la alte poziții și se compară **similaritatea cosinus** între vectorul proiectat și vectorii aleși. Astfel, funcția de pierdere contrastivă încurajează similaritatea cu exemplele *adevărat pozitive* și penalizează similaritatea cu exemplele *negative*.

De asemenea, în timpul preantrenării, se folosește și o funcție de pierdere pentru diversitatea *codewords*, numită *Diversity Loss*, care maximizează entropia distribuției *Gumbel-Softmax*. Această funcție impune modelului să exploreze mai multe sunete fonetice în timpul antrenării și previne blocarea pe un subset de sunete.

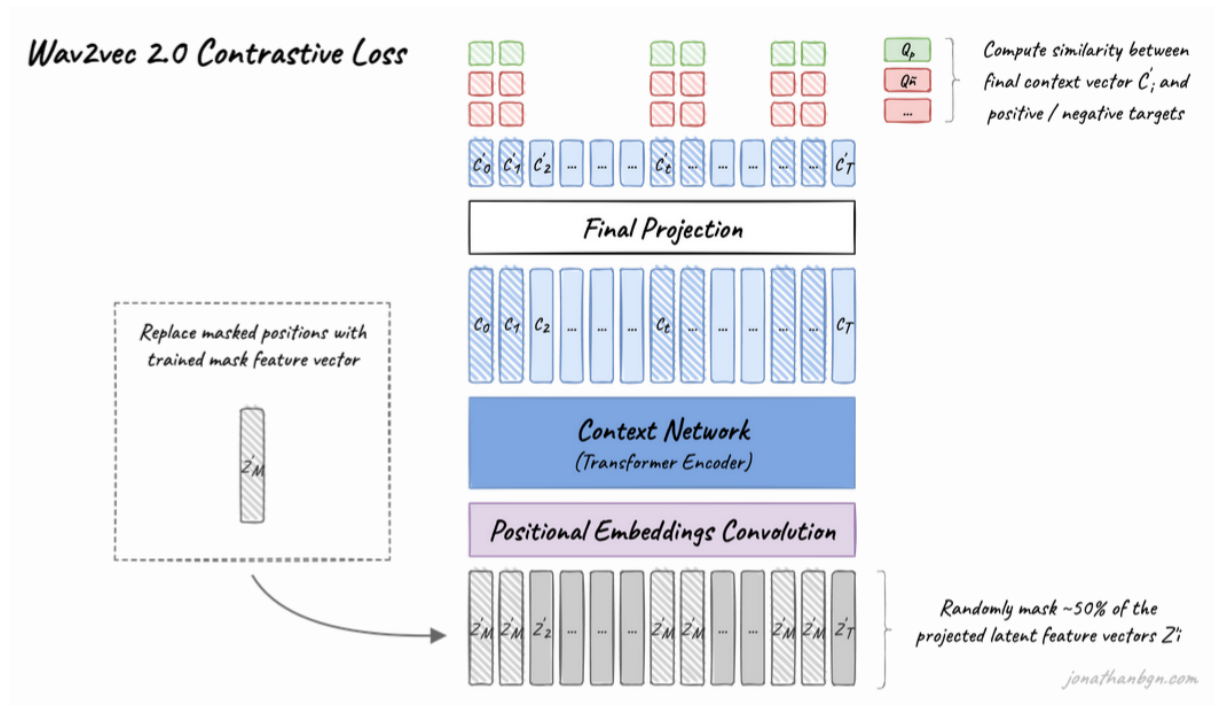


Figura 2.7: Arhitectura componentei *Contrastive Loss*<sup>1</sup>

<sup>1</sup>Imaginile au fost preluate de pe site-ul lui Jonathan Bgn, "Illustrated Wav2Vec 2.0", disponibil la: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>.

### 2.1.3 Setul de date

Modelul oficial a fost preantrenat pe setul de date *LibriSpeech*, iar eu am continuat antrenarea pe seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1*.

#### Mini-LibriSpeech

*Mini LibriSpeech* este un subset al setului de date *LibriSpeech* care conține aproximativ 2 ore de înregistrări audio la o frecvență de eșantionare de 16 kHz. În medie, fiecare înregistrare are o durată de 6.72 secunde, cel mai lung audio având o durată de 31.5 secunde.

#### Common Voice Delta Segment 16.1

*Common Voice Delta Segment 16.1* este un subset al setului de date *Common Voice* care conține aproximativ 2 ore de înregistrări audio la o frecvență de eșantionare de 48 kHz. A fost nevoie să reducem frecvența de eșantionare la 16 kHz pentru a putea folosi aceste date la antrenarea modelului. În medie, fiecare înregistrare are o durată de 5.63 secunde, cel mai lung audio având o durată de 10.47 secunde.

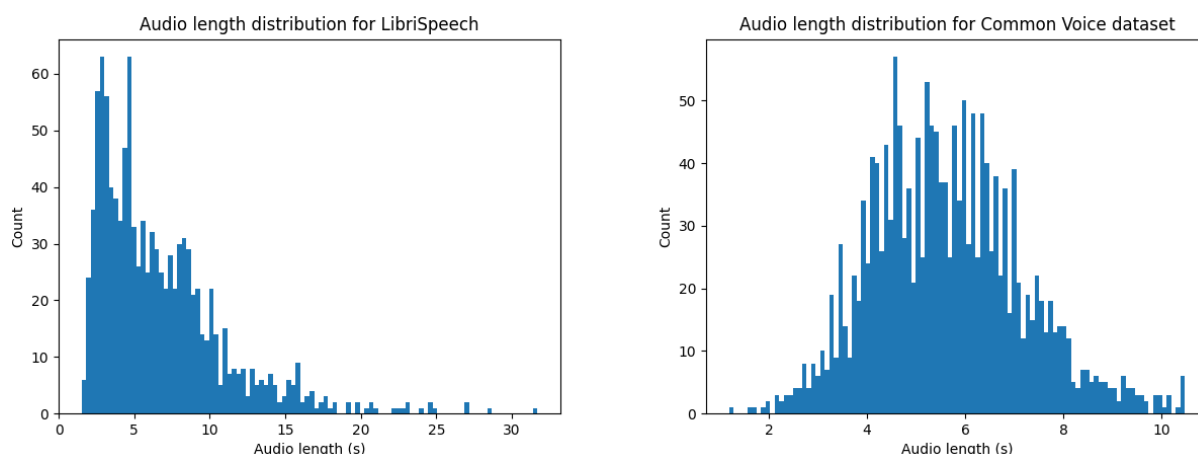


Figura 2.8: Distribuția duratelor secvențelor audio din seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1*

### Concluzie

Menționăm aceste detalii deoarece pentru generarea subtitrărilor vom avea nevoie de secvențe audio mult mai lungi decât cele folosite pentru antrenare, care nu ar încăpea în memorie. Astfel, va trebui să folosim o tehnică de segmentare a secvențelor audio în bucăți mai mici pentru a putea fi procesate. Mai multe detalii despre această tehnică vor fi prezentate în secțiunea *Subtitrări*. 3.3.3

### 2.1.4 Antrenarea modelului

Pentru antrenarea modelului am folosit limbajul de programare *Python* și biblioteca *Hugging Face* care pune la dispoziție o serie de librării precum *transformers* și *datasets*.

Hiperparametrii folosiți pentru antrenarea modelului sunt:

- **Rata de învățare** - pentru a controla cât de mult se vor actualiza ponderile în timpul antrenării, poate fi gândit ca lungimea pasului în direcția gradientului
- **Declinul ponderilor** - pentru a menține sub control creșterea ponderilor în timpul antrenării și pentru a evita gradientii instabili
- **Pași de încălzire** - pentru a controla cum se modifică rata de învățare în prima parte a antrenării
- **Dimensiunea lotului** - câte exemple se procesează în același timp

Tabela 2.1: Hiperparametrii folosiți pentru antrenarea modelului *wav2vec2*

Hiperparametru	Valoare
Rata de învățare	$1 \times 10^{-4}$
Declinul ponderilor	0.005
Pași de încălzire	1000
Dimensiunea lotului	4

Am antrenat modelul pe seturile de date *Mini LibriSpeech* și *Common Voice Delta Segment 16.1* pe o placă grafică *NVIDIA Tesla V100* pusă la dispoziție de *Google Colab*. Am salvat starea modelului la fiecare 500 de pași pentru a putea monitoriza evoluția modelului. Rezultatele obținute la fiecare punct de control *facebook/wav2vec2-base* sunt prezentate în cele două tabele de mai jos.

#### Notă

Se observă că modelul a început să învețe destul de repede, scăzând pierderea de antrenare de la 3.840 la 0.066, respectiv de la 3.919 la 0.064 în doar 7000 de pași. Urmărind graficul, se observă fenomenul de *overfitting* care apare în jurul pașilor 5000-6000, motiv pentru care am ales să opresc antrenarea la 7000 de pași și să folosesc modelul de la pasul 5000, respectiv 5500.

### 2.1.5 Îmbunătățire cu un model de limbaj

Pentru a îmbunătăți recunoașterea vorbirii, am folosit un model de limbaj bazat pe n-gramă care mărește performanța modelului de la **wer 4.2%** la **wer 2.9%**, aducând o îmbunătățire de **1.3%**.

Tabela 2.2: *Mini LibriSpeech*

Pas	Antrenare	Validare
500	3.840	3.099
1000	1.202	0.586
1500	0.360	0.352
2000	0.231	0.333
2500	0.163	0.357
3000	0.136	0.331
3500	0.114	0.369
4000	0.104	0.348
4500	0.094	0.335
5000	0.083	0.284
5500	0.078	0.332
6000	0.072	0.356
6500	0.069	0.393
7000	0.066	0.380

Tabela 2.3: *Common Voice Delta 16.1*

Pas	Antrenare	Validare
500	3.919	3.236
1000	1.287	0.570
1500	0.368	0.400
2000	0.226	0.371
2500	0.166	0.402
3000	0.136	0.475
3500	0.116	0.445
4000	0.097	0.448
4500	0.096	0.404
5000	0.079	0.459
5500	0.080	0.400
6000	0.069	0.445
6500	0.073	0.421
7000	0.064	0.440

### Model de limbaj bazat pe n-grame

Un un model de limbaj bazat pe n-grame este un model statistic care estimează, în cazul nostru, probabilitatea apariției unui caracter având în vedere cele n-1 caractere anterioare. Modelul se bazează pe ipoteza **Markov de ordinul n**, conform căreia putem aproxima probabilitatea apariției unui caracter folosind doar ultimele n caractere. Formula de mai jos ilustrează această idee:

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^m P(w_i | w_1, w_2, \dots, w_{i-1}) \approx \prod_{i=1}^m P(w_i | w_{i-(n-1)}, \dots, w_{i-1}) \quad (2.1)$$

Am folosit un context de 5 caractere și am antrenat modelul pe setul de date *Helsinki-NLP/europarl* [13] deoarece conține și texte în limba engleză și putem avea certitudinea că textele sunt corecte din punct de vedere gramatical. Cu ajutorul librăriei *KenLM* [8], am antrenat modelul de n-grame și apoi am creat un procesor specific Hugging Face pentru modelele *wav2vec 2.0*.

În mod normal, modelul *wav2vec 2.0* ia argumentul maxim din distribuția de probabilitate pentru a prezice caracterul următor. În schimb, cu ajutorul unui un model de limbaj bazat pe n-grame, aceste probabilități sunt alterate pentru a se apropia de limbajul natural.



## Îmbunătățire cu un model de limbaj bazat pe arhitectura Transformer

Având în vedere performanța arhitecturii Transformer în învățarea secvențelor, o altă abordare ar fi utilizarea unui model de limbaj bazat pe Transformer. Paper-ul original [2] menționează în Appendix-ul C că un model de limbaj bazat pe Transformer este într-adevăr mai bun decât un model de limbaj bazat pe n-grame, dar durata antrenării și a inferenței este mult mai mare.

Tabelul de mai jos a fost extras din paper-ul original și prezintă rezultatele obținute.

Model	Unlabeled data	LM	dev		test	
			clean	other	clean	other
LARGE - from scratch	-	None	2.8	7.6	3.0	7.7
	-	4-gram	1.8	5.4	2.6	5.8
	-	Transf.	1.7	4.3	2.1	4.6
BASE	LS-960	None	3.2	8.9	3.4	8.5
		4-gram	2.0	5.9	2.6	6.1
		Transf.	1.8	4.7	2.1	4.8
LARGE	LS-960	None	2.6	6.5	2.8	6.3
		4-gram	1.7	4.6	2.3	5.0
		Transf.	1.7	3.9	2.0	4.1
LARGE	LV-60k	None	2.1	4.5	2.2	4.5
		4-gram	1.4	3.5	2.0	3.6
		Transf.	1.6	3.0	1.8	3.3

Figura 2.9: Rezultatele obținute cu un model simplu, un model de limbaj bazat pe n-grame și un model de limbaj bazat pe Transformer

Deoarece modelul de limbaj bazat pe *Transformer* necesită considerabil mai mult timp pentru antrenare și inferență, iar îmbunătățirea pe care o aduce nu este cu mult mai semnificativă decât cea adusă de un model de limbaj bazat pe n-grame, am ales să folosesc doar modelul îmbunătățit cu n-grame.

## 2.2 Clasificarea videoclipurilor

Videoclipurile încărcate pe platform vin însoțite de metadate precum: titlu, descriere și, folosind modelul prezentat anterior, subtitrări. Pentru o experiență mai personalizată, am ales să clasific videoclipurile în funcție de subiectul abordat în topicuri precum: politică, sport, divertisment, tehnologie și afaceri. Am folosit un modelul de clasificare **BERT** (Bidirectional Encoder Representations from Transformers) dezvoltat de *Google* [4] pe care l-am antrenat pe setul de date *BBC News* [7].

### 2.2.1 Arhitectura modelului

Modelul *BERT* este un model de învățare profundă care are la bază partea de encodare a unui *Transformer* [14], scopul lui fiind de a înțelege contextul cuvintelor într-o propoziție. Modelul *BERT* vine în două variante: *BERT-base* și *BERT-large*, cele două diferă prin numărul de blocuri de encodare (12 și 24), numărul de neuroni din stratul pentru clasificare (*fully-connected*) (768 și 1024) și numărul de capete de atenție (*attention heads*) (12 și 16).

Spre deosebire de arhitectura standard a unui *Transformer*, *BERT* adaugă un token special la începutul fiecărei propoziții, *[CLS]*, folosit pentru clasificare. Intuitiv, acest token va reține informații despre întreaga propoziție și va putea fi folosit pentru rețeaua de clasificare care va determina topicul propoziției. 2.10

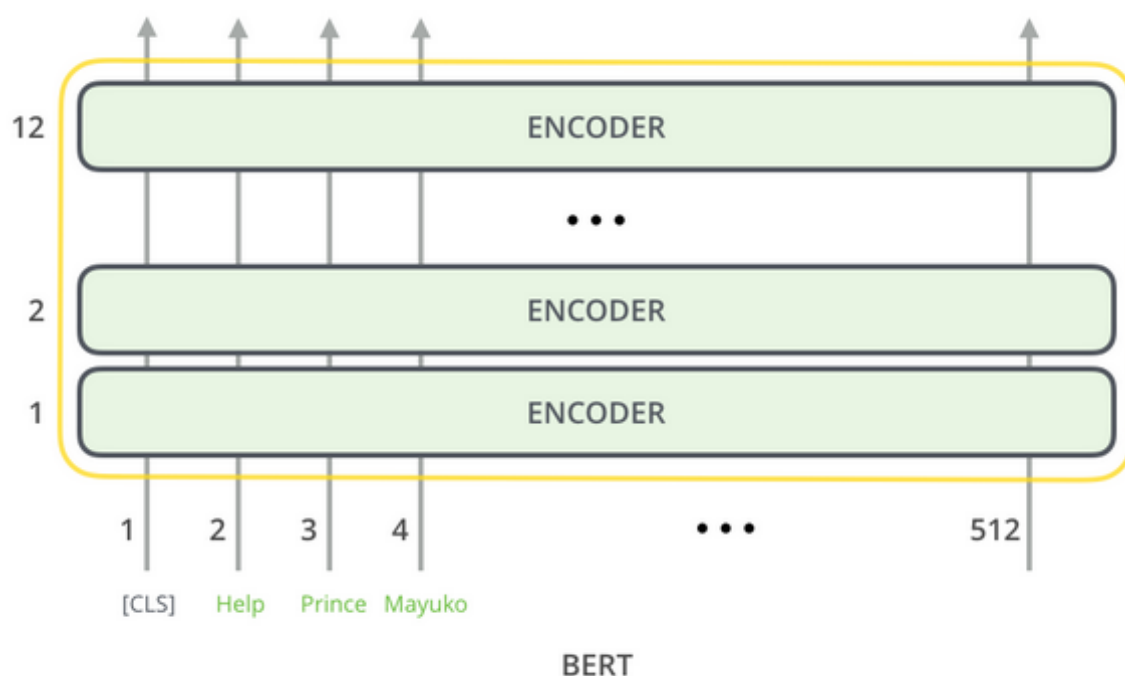


Figura 2.10: Arhitectura modelului *BERT* <sup>2</sup>

Ieșirile modelului *BERT* au dimensiunea 768, iar pentru partea de clasificare sunt trecute prin un strat liniar (*fully-connected*) cu 5 neuroni, câte unul pentru fiecare clasă. Cu ajutorul funcției *softmax*, se obține o distribuție de probabilitate peste cele 5 clase, iar clasa cu cea mai mare probabilitate este cea aleasă.

<sup>2</sup>Imaginea a fost preluată de pe site-ul lui Jay Alammar, "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)", disponibil la: <https://jalammar.github.io/illustrated-bert/>.

## 2.2.2 Setul de date

Pentru antrenarea modelului am folosit setul de date *BBC News* care conține 2225 de articole între anii 2004-2005. Setul de date este împărțit în 5 clase: politică, sport, divertisment, tehnologie și afaceri. Mai jos am prezentat distribuția datelor din setul de date din punct de vedere al exemplurilor pe clasă și al lungimii medii a articolelor. 2.11

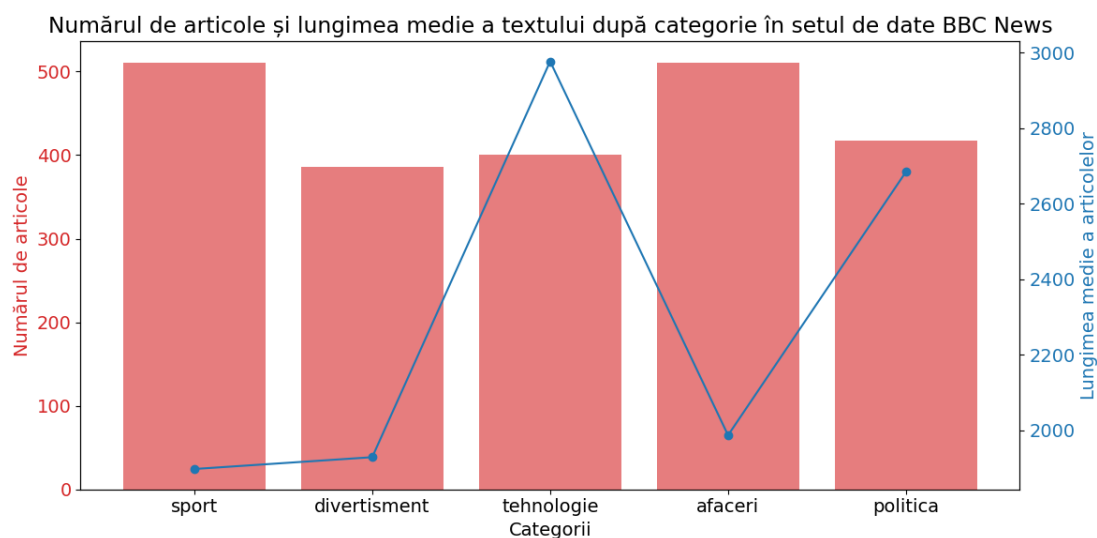


Figura 2.11: Distribuția claselor din setul de date *BBC News*

## 2.2.3 Antrenarea modelului

Antrenarea modelului a fost făcută folosind limbajul de programare Python, biblioteca PyTorch [11] și plecând de la modelul preantrenat *bert-base-uncased* pus la dispoziție de Hugging Face. Codul a fost structurat în:

- **dataset.py** - creează un Dataset specific PyTorch pentru setul de date *BBC News*, implementând metodele `__len__` și `__getitem__`; metoda `__getitem__` aplică tokenizer-ul pe text cu lungimea maximă de 512 token-uri și returnează un tuplu de `input_ids`, `attention_mask` și `label`.
- **model.py** - definește arhitectura modelului *BERT* și a rețelei de clasificare: inițializează modelul preantrenat *bert-base-uncased*, extrage token-ul special `[CLS]` și adaugă un strat liniar cu 5 neuroni pentru clasificare.
- **trainer.py** - antrenează modelul pe setul de date *BBC News* folosind hiperparametrii primiți din linia de comandă și salvează modelul de fiecare dată când obține o acuratețe mai bună pe setul de validare.

În figurile de mai jos 2.12 sunt ilustrate pierderile de antrenare și de evaluare pentru modelul *BERT* folosind ca rată de învățare  $1 \times 10^{-4}$  și  $1 \times 10^{-5}$ . Se observă fenomenul de *overfitting* pentru rata de învățare  $1 \times 10^{-4}$ , motiv pentru care am ales să experimentez cu valori mai mici.

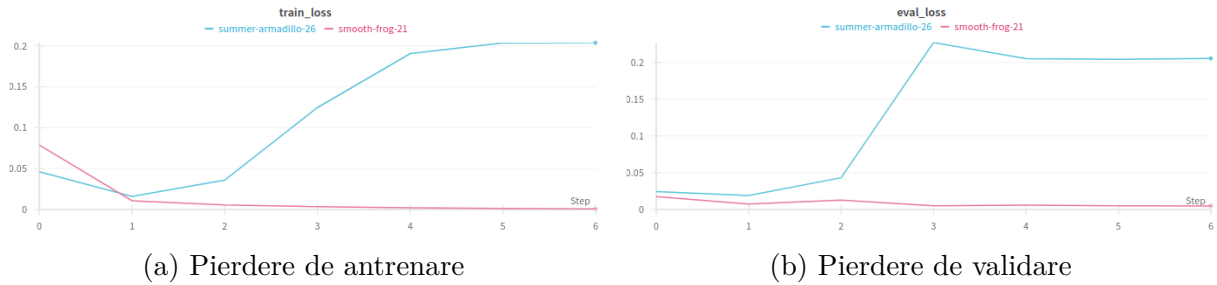


Figura 2.12: Pierderile de antrenare și de validare pentru modelul *BERT*

Am încercat mai multe experimente plecând de la aceiași hiperparametri, cu rata de învățare  $1 \times 10^{-5}$ , pentru a vedea cum e comportă modelul la diferite inițializări ale ponderilor. În figura 2.13 se observă că acuratețile obținute sunt în intervalul 0.95-0.99. Am ales, în final, să folosesc modelul cu cele mai mari valori obținute pe setul de validare.



Figura 2.13: Antrenamentele modelului *BERT* cu rata de învățare  $1 \times 10^{-5}$

## Hiperparametrii

Pentru modelul final pe care l-am folosit pentru clasificarea videoclipurilor am ales următorii hiperparametrii:

Hiperparametru	Valoare
Rată de învățare	0.00001
Dimensiunea lotului	8
Optimizator	Adam
Funcția de pierdere	CrossEntropyLoss

Tabela 2.4: Hiperparametrii modelului *BERT*

## 2.2.4 Îmbunătățirea clasificării

Videoclipurile pe care le vom clasifica mai departe conțin informații precum: titlu, descriere și subtitrări. O primă observație este că titlul și descrierea sunt de obicei scurte și se poate deduce topicul videoclipului doar din aceste două informații. Subtitrările, pe de altă parte, sunt mai lungi și conțin mai multe informații irelevante pentru clasificare.

În acest sens, textul alcătuit din titlu, descriere și subtitrări va fi împărțit în blocuri de câte 512 token-uri pentru a respecta restricția de intrare a modelului *BERT*. Pentru a exploata observația de mai sus, vom folosi doar primele 10 blocuri de text pentru a clasifica videoclipul și vom da o importanță mai mare în votul final primului bloc de text, care conține titlul și descrierea videoclipului.

Pentru rezultatul final, vom alege topicul cu frecvența majoritară din cele 10 blocuri de text.

## 2.3 Concluzie

Cele două modele prezentate anterior, *wav2vec 2.0* și *BERT*, au fost încărcate pe platforma *Hugging Face* și sunt disponibile în pachetul *transformers*.

### wav2vec 2.0

Utilizarea modelului de recunoaștere a vorbirii necesită folosirea modulului *Wav2Vec2ForCTC* și a modulului *Wav2Vec2Processor*, respectiv *Wav2Vec2ProcessorWithLM* pentru a folosi varianta îmbunătățită cu n-grame. Modelele se găsesc pe *Hugging Face* și se pot accesa din pachetul *transformers* astfel:

- *Mini LibriSpeech* - *3funnn/wav2vec2-base-minilibrispeech*, respectiv *3funnn/wav2vec2-base-minilibrispeech-lm* pentru modelul îmbunătățit cu n-grame
- *Common Voice Delta Segment 16.1* - *3funnn/wav2vec2-base-common-voice*, respectiv *3funnn/wav2vec2-base-common-voice-lm* pentru modelul îmbunătățit cu n-grame

### BERT

Modelul de clasificare a videoclipurilor este disponibil în pachetul *transformers* sub numele *3funnn/bert-topic-classification*. Modelul primește ca intrare un text (care poate fi o concatenare a titlului și a descrierii videoclipului) și returnează o distribuție de probabilitate pentru fiecare din cele 5 clase. *Hugging Face* pune la dispoziție funcția *pipeline* care permite folosirea modelului într-un mod simplu.

# Capitolul 3

## Prezentarea aplicației

### 3.1 Arhitectura aplicației

Am ales să folosesc arhitectura **MERN** (MongoDB, Express.js, React, Node.js), la care am mai adăugat două servere de **Flask** pentru recunoașterea vorbirii și clasificarea videoclipurilor, o bază de date **Elasticsearch** pentru căutare și Kibana pentru vizualizarea datelor din Elasticsearch.

În figura de mai jos (Fig. 3.1) este prezentată arhitectura aplicației:

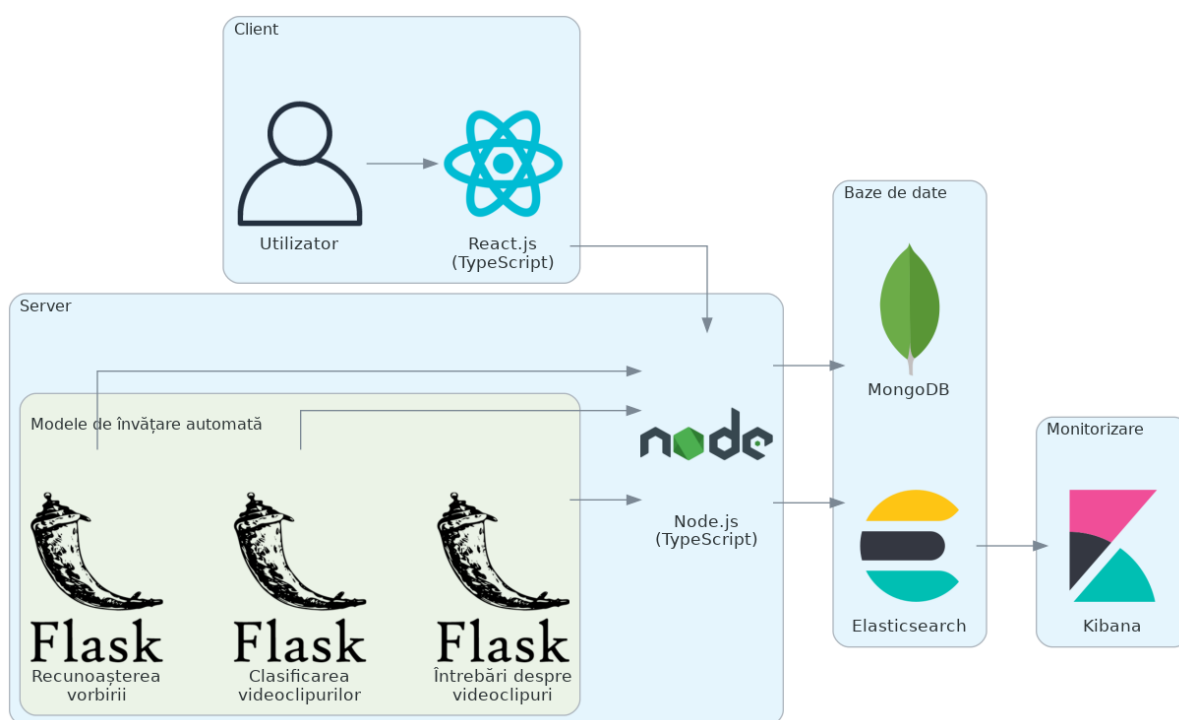


Figura 3.1: Arhitectura aplicației

## 3.2 Frontend

Am folosit React și limbajul de programare TypeScript pentru interfața grafică a aplicației web.

**React.js** este o librărie JavaScript care simplifică procesul de dezvoltare a aplicațiilor web prin caracteristici specifice precum reutilizarea componentelor și actualizarea independentă a elementelor interfeței grafice. **TypeScript** este un limbaj de programare creat de Microsoft care extinde limbajului JavaScript cu tipuri statice, aspect util care îmbunătățește calitatea codului.

### 3.2.1 React

Librăria React impune o serie de concepte și bune practici care abstractizează și modularizează codul aplicației web. Dintre acestea, cele mai importante sunt:

- **Components:** Ideea de bază a librăriei React o reprezintă componentele, elemente reutilizabile care încapsulează și izolează logica codului.
- **Props:** Proprietățile sunt parametri transmiși de la componenta părinte la componenta copil, fiind folosite pentru customizarea comportamentului componentelor.
- **State:** Starea reprezintă datele interne ale componentelor, fiind folosite pentru actualizarea interfeței grafice din JSX.
- **Hooks:** Hooks sunt funcții speciale care permit interacțiunea cu starea și ciclul de viață al componentelor, fiind succesorul claselor din versiunile mai vechi ale librăriei.
- **Contexts:** Contextele sunt o modalitate de a controla transmiterea datelor între componente fără a folosi proprietăți, intuitiv, creând un fel de scope pentru componentele din interiorul contextului.
- **Models:** Deoarece folosim TypeScript, avem nevoie de modele pentru a defini tipurile de date folosite în aplicație.
- **Views/Pages:** Pot fi considerate componente principale, fiind rutele aplicației web.
- **Services:** Serviciile sunt funcții asincrone care primesc date din frontend, creează cereri HTTP către server (folosind *Axios* sau *Fetch*) și returnează răspunsurile primite de la server.
- **Utils:** Funcții utilitare care nu sunt legate de o componentă anume.

### 3.2.2 Autentificare

Autentificarea utilizatorilor nu folosește niciun serviciu extern, ci se bazează pe un sistem propriu de înregistrare și logare. Fiecărei cereri de la client i se atașează un token JWT (JSON Web Token) generat de server la logare/înregistrare, fiind folosit pentru a verifica identitatea utilizatorului.

#### JWT

JSON Web Token (JWT) encodează în structura sa 3 componente: *header*, *payload* și *signature*. *Header*-ul stochează tipul de token și algoritmul de criptare, *payload*-ul conține informații despre utilizator precum numele și data emiterii, iar *signature*-ul este un hash al primelor două componente la care se mai adaugă un secret.

#### Înregistrare și logare

Atât înregistrarea 3.2, cât și logarea 3.3 sunt realizate prin intermediul unor formulare care cer informații precum numele, prenumele, adresa de email, numele de utilizator, parola și poza de profil. Pentru aceste formulare au fost folosite validatoare care verifică dacă datele introduse sunt corecte.

Odată ce datele sunt introduse, sunt validate încă o dată la nivel de server pentru a evita eventualele atacuri și, dacă sunt corecte, se generează un token JWT care este trimis înapoi la client. Acest token este salvat în memoria locală a browser-ului, dar și în contextul utilizatorului pentru a fi folosit în cererile ulterioare către server.

Pentru deconectare, token-ul este șters din memoria locală și contextul utilizatorului.


### 3.2.3 Profil

Pagina de profil prezintă informații despre utilizator precum:

- **Date despre utilizator:** nume, prenume, nume de utilizator și poza de profil.
- **Date despre interacțiunea cu platforma:** numărul de videoclipuri încărcate, numărul de utilizatori urmăriți și numărul de utilizatori care îl urmăresc.
- **Postări:** postările utilizatorului pe platformă sub forma unui grid.
- **Urmăritori și urmăriți:** vizibile în *Modal*-uri separate.
- **Setări:** posibilitatea de a-și edita datele personale.

Când un alt utilizator îi accesează profilul, acesta îl poate adăuga sau scoate din lista de urmăriți, poate începe o conversație cu acesta sau poate vizualiza postările acestuia. (Fig. 3.4)





## Register

First Name \*

Last Name \*


Email Address \*

Username \*

Password \*

**SIGN UP**

[Already have an account? Sign in](#)



## Login

Username \*

Password \*


**LOG IN**

[Don't have an account? Sign up](#)

Figura 3.3: Pagina de logare

Figura 3.2: Pagina de înregistrare

De asemenea, poate vizualiza listele menționate anterior de urmăritori și urmăriți cu ajutorul unui *Modal* care se deschide atunci când se apasă pe numărul de urmăritori/urmăriți. De aici, poate naviga mai departe la profilul utilizatorilor respectivi.




**cristina.trifan**

Cristina Trifan

3 posts      1 followers      1 following


**UNFOLLOW**   **CHAT**

Videos by cristina.trifan




The quick proof of Baye...

Cristina Trifan



Peter Drury talks about ...

Cristina Trifan



Everything you should r...

Cristina Trifan

Figura 3.4: Pagina de profil

### 3.2.4 Video

Platforma permite utilizatorilor să încarce videoclipuri, să le vizualizeze și să interacționeze cu acestea prin intermediul unor funcții precum aprecieri și comentarii.

## Încărcare videoclip

Încărcarea videoclipurilor se face printr-un buton care redirecționează utilizatorul către pagina de încărcare. Aici, utilizatorul poate selecta un fișier video, adăuga un titlu și o descriere, precum și hashtags care descriu conținutul videoclipului. După ce videoclipul este încărcat, începe un proces de salvare a metadatelor în baza de date, de salvare a videoclipului pe mașina serverului, de extragere și recunoaștere a vorbirii și de generare a subtitrărilor, precum și de clasificare a videoclipului într-un anumit topic. (Fig. 3.5)

## Editare

Autorii videoclipurilor încărcate pot accesa pagina de editare a videoclipului prin butonul de editare de pe pagina de vizualizare. Fiind o rută protejată, utilizatorul trebuie să fie autentificat și să fie autorul videoclipului pentru a putea schimba metadatele videoclipului. (Fig. 3.6)

Upload Video

Title \*  
New video from 3blue1brown

Description \*  
time for science

Video File \*  
The quick proof of Bayes' theorem.mp4 +

Add Hashtags (press Enter to add)  
#proof

Type a hashtag and press Enter

#science #math #theorem

UPLOAD

Figura 3.5: Încărcare videoclip

Edit Video

Title \*  
Banking Explained | Money and Credit

Description \*  
video about money

Add Hashtags (press Enter to add)  
#credit

Type a hashtag and press Enter

#money #bank #explained

UPDATE

DELETE

Figura 3.6: Editare videoclip

## Pagina principală

Pagina principală (Fig. 3.7) a aplicației conține următoarele elemente:

- **Bara de căutare:** permite utilizatorului căutarea videoclipurilor după cuvintele cheie introduse în bara de căutare.
- **Butonul de încărcare:** redirecționează către pagina de încărcare a videoclipurilor.
- **Grid de videoclipuri:** afișează toate videoclipurile încărcate de utilizatori pe platformă. Fiecare videoclip conține o imagine reprezentativă, titlul și numele

autorului și o bară colorată care reprezintă topicul videoclipului. Accesarea acestuia redirecționează utilizatorul către pagina de vizualizare.

- **Filtre și sortare:** utilizatorul poate filtra videoclipurile după filtre precum: topic și intervalul de popularitate (numărul de vizualizări) și le poate sorta după criterii precum: nou-vechi, vechi-nou, popularitate-crescător, popularitate-descrescător.

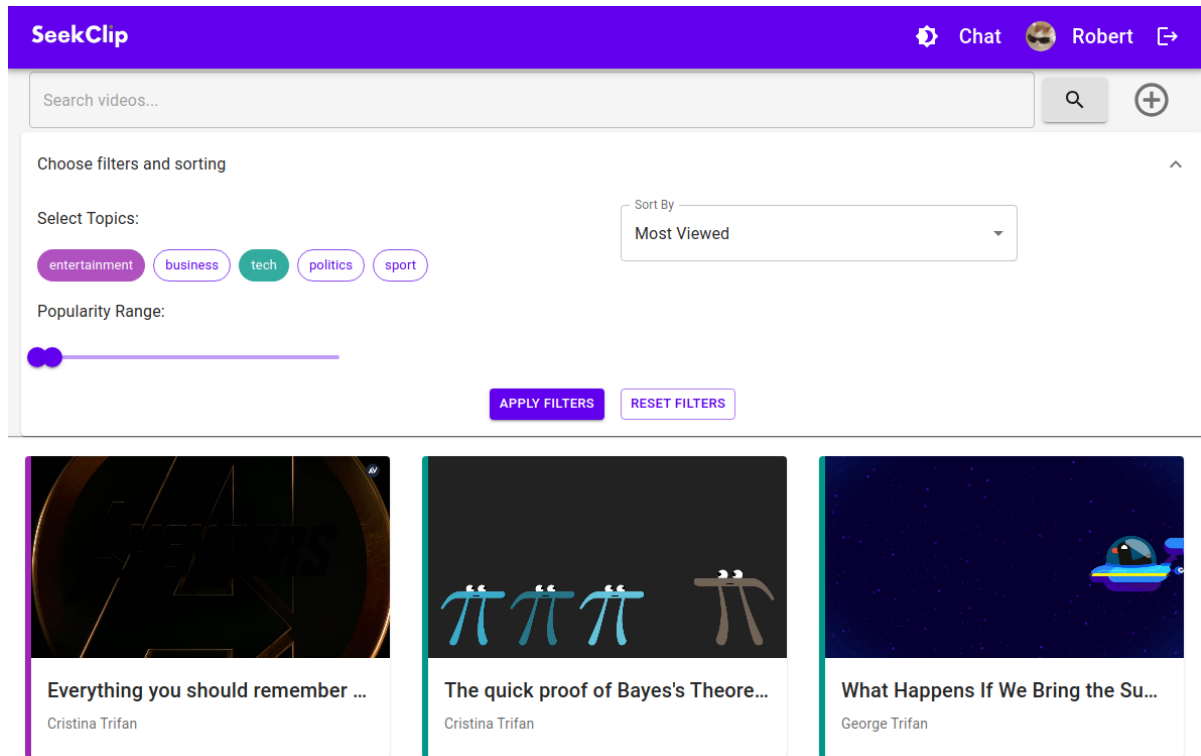


Figura 3.7: Grid de videoclipuri

## Vizualizarea unui videoclip

videoclipurile încărcate sunt afișate pe pagina principală a aplicației web sub forma unui grid, fiecare videoclip având o imagine reprezentativă pentru acesta, un titlu, numele autorului și o bară colorată, sugestivă pentru topicul videoclipului.

Atunci când este accesat un videoclip, utilizatorul este redirecționat către o pagină dedicată (Fig. 3.8) unde poate interacționa cu videoclipul astfel:

- **Vizualizare:** videoclipul este afișat într-o fereastră video care permite alegerea subtitrărilor, schimbarea volumului și manipularea videoclipului.
- **Editare:** editarea videoclipului poate fi accesată doar de autorul acestuia și permite schimbarea informațiilor despre videoclip. (titlu, descriere, hashtags)

- **Informații:** titlul și descrierea videoclipului sunt afișate sub fereastra menționată anterior, descrierea fiind accesată printr-un buton de expandare într-o componentă acordeon.
- **Autor:** numele și poza de profil a autorului videoclipului permit redirecționarea către profilul acestuia.
- **Aprecieri:** utilizatorul poate da aprecia sau nu videoclipul apăsând pe iconițele dedicate în acest sens.
- **Distribuire:** videoclipul poate fi împărtășit cu utilizatorii platformei prin intermediul unui buton de distribuire care deschide un *Modal* și listează persoanele urmărite de utilizator.
- **Vizualizări:** numărul de accesări ale videoclipului este afișat sub numărul de aprecieri.
- **Comentarii:** lista de comentarii este afișată sub formă de ierarhie, unde comentariile pot fi fie răspunsuri la alte comentarii, fie comentarii directe la videoclip.
- **Adăugare comentariu:** utilizatorul poate adăuga și el la rândul lui comentarii la videoclip sau răspunde la alte comentarii.
- **Editare comentarii:** doar autorii comentariilor își pot edita sau șterge propriile comentarii.
- **Întrebări:** utilizatorul poate pune întrebări referitoare la conținutul videoclipului prin intermediul unui *Modal*. În spatele acestui *Modal* se află un serviciu care folosește subtitrările videoclipului și accesează API-ul de la ChatGPT pentru a genera răspunsuri.
- **Hashtags:** lista de hashtag-uri afișată sub descrierea videoclipului oferă mai multă expresivitate pe lângă topicurile alese anterior.

## Aprecieri & Comentarii

După cum am menționat anterior, pagina de vizualizare permite utilizatorului să aprecieze sau nu videoclipul și să adauge comentarii. Fiecare videoclip reține o listă cu id-urile utilizatorilor care au dat și-au exprimat aprecierea pentru și o listă cu comentariile adăugate. De asemenea, utilizatorii pot răspunde la comentarii, creând astfel o ierarhie de comentarii și pot edita sau șterge comentariile proprii.

Dacă un comentariu are răspunsuri, atunci textul devine șters *[deleted]*, fiind un *soft-delete*. Altfel, dacă un comentariu nu are răspunsuri, atunci acesta este șters complet

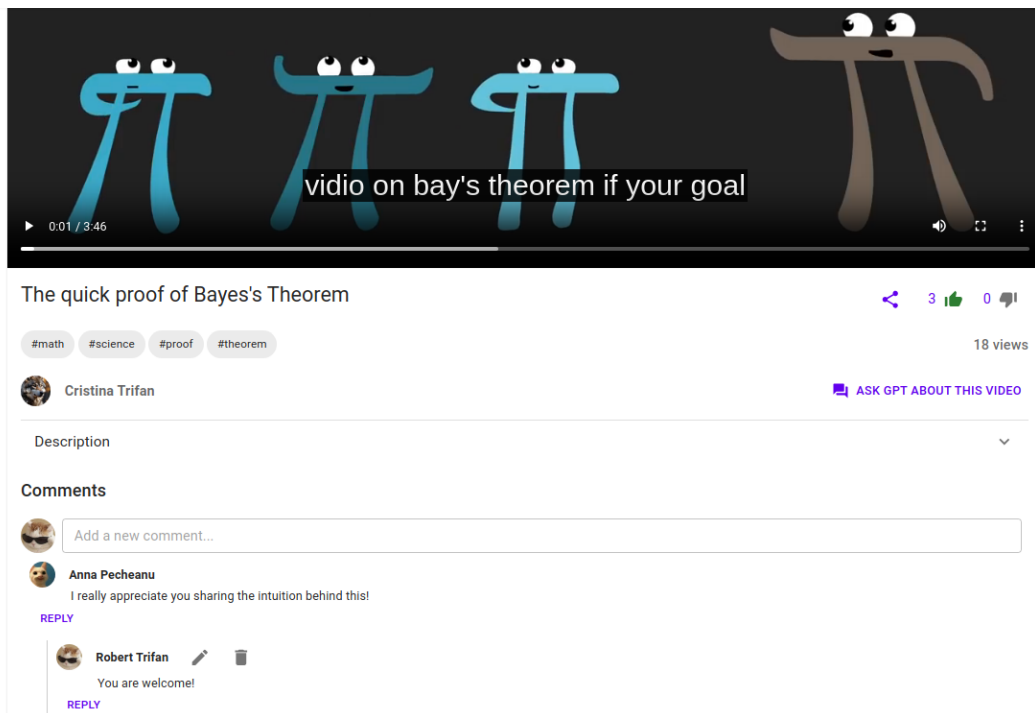


Figura 3.8: Pagina de vizualizare a unui videoclip

din baza de date, fiind considerat un *hard-delete*. Comentariile sunt sortate după data adăugării, cele mai recente fiind afișate mai sus. (Fig. 3.9)

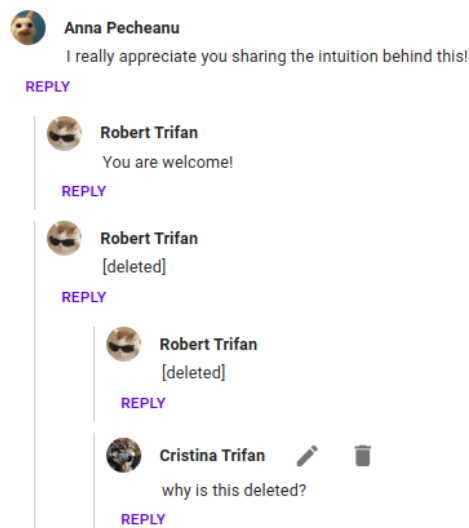


Figura 3.9: Secțiunea de comentarii

Trebuie să menționăm un detaliu important legat de ierarhia comentariilor. Ar fi foarte costisitor, din punct de vedere al performanței, ca de fiecare dată când se șterge un comentariu să se reconstruiască arborele de comentarii. De aceea, am creat un serviciu care rulează o dată pe zi, construiește ierarhia comentariilor și șterge acele drumuri în arbore care nu mai sunt folosite. Mai multe detalii despre acest serviciu se găsesc în secțiunea 3.5.1.

### 3.2.5 Căutare

#### Bara de căutare

Pe pagina principală a aplicației web se află o bară de căutare care permite utilizatorului introducerea unor cuvinte cheie specifice videoclipurilor căutate. De asemenea, poate filtra și sorta rezultatele obținute după criteriile menționate anterior. Am adăugat aici încă două criterii: relevanță-descrescător și relevanță-crescător, care sortează rezultatele după scorurile obținute de Elasticsearch. (Fig. 3.10)

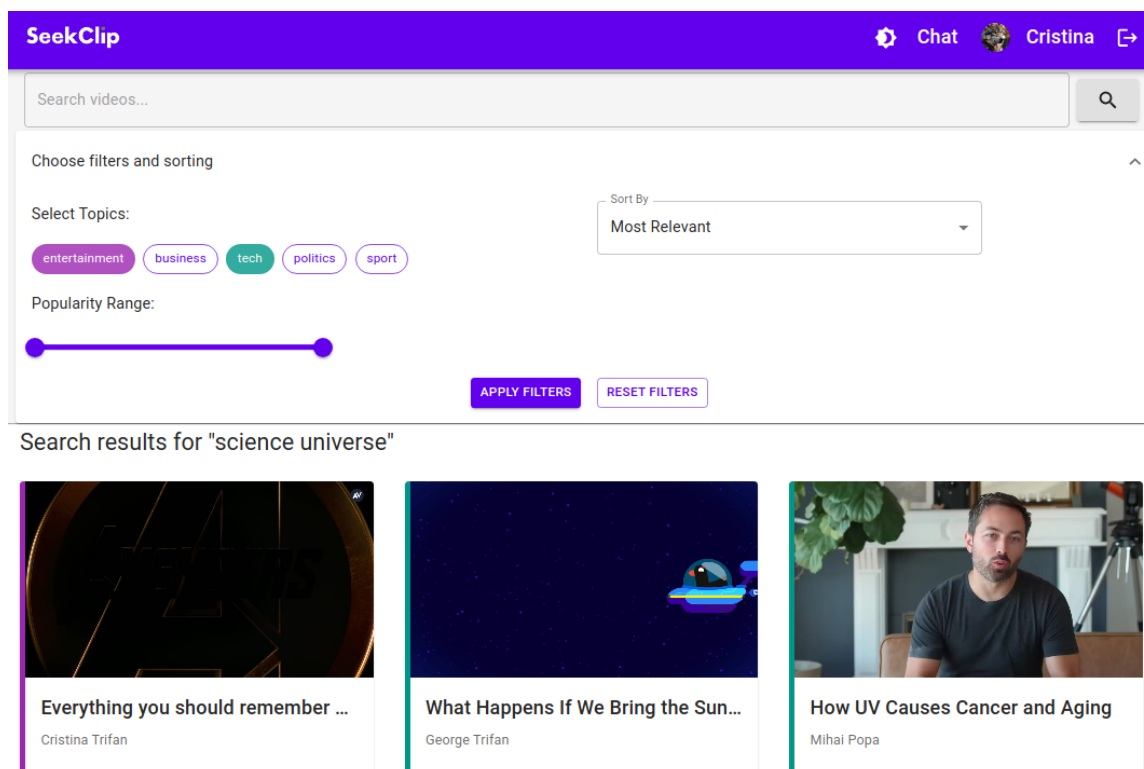


Figura 3.10: Bară de căutare

Când utilizatorul introduce textul în bara de căutare, aplicația web trimite o cerere către server care caută apoi videoclipurile într-o bază de date Elasticsearch. Datorită parametrului de *confuzie* (*fuzziness*), căutarea este tolerantă la greșeli de scriere și oferă rezultate relevante.

### 3.2.6 Conversație

Platforma oferă utilizatorilor posibilitatea de a comunica între ei în timp real prin mesaje private. Serviciul de mesagerie are la bază WebSocket, un protocol de comunicare bidirecțional între client și server care permite emiterea și ascultarea de evenimente. Practic, serverul joacă rolul de intermediar în conversația dintre doi utilizatori, în sensul că primește mesajele de la un utilizator și le trimite mai departe către celălalt utilizator cu care conversează. (Fig. 3.11)

Utilizatorul poate accesa pagina de conversație din bara de navigare, unde îi sunt afișați utilizatorii pe care îi urmărește și statusul lor (activi/inactivi), dar și restul utilizatorilor de pe platformă activi. Aici poate citi mesajele anterioare, poate trimite/primi mesaje în timp real și poate vedea când un utilizator este activ/inactiv.

De asemenea, poate distribui videoclipurile de pe platformă cu alți utilizatori folosind butonul de distribuire, care deschide un *Modal* cu lista de utilizatori urmăriți.

Inițial, utilizatorul emite un eveniment de conectare către server, iar serverul îl salvează într-un dicționar de utilizatori conectați. Apoi, având un canal bidirecțional deschis, utilizatorul poate emite și asculta evenimente de la server. În cazul în care utilizatorul se deconectează, serverul închide canalul și îl șterge din dicționarul de utilizatori activi. React pune la dispoziție *useEffect* pentru a gestiona evenimentele menționate anterior.

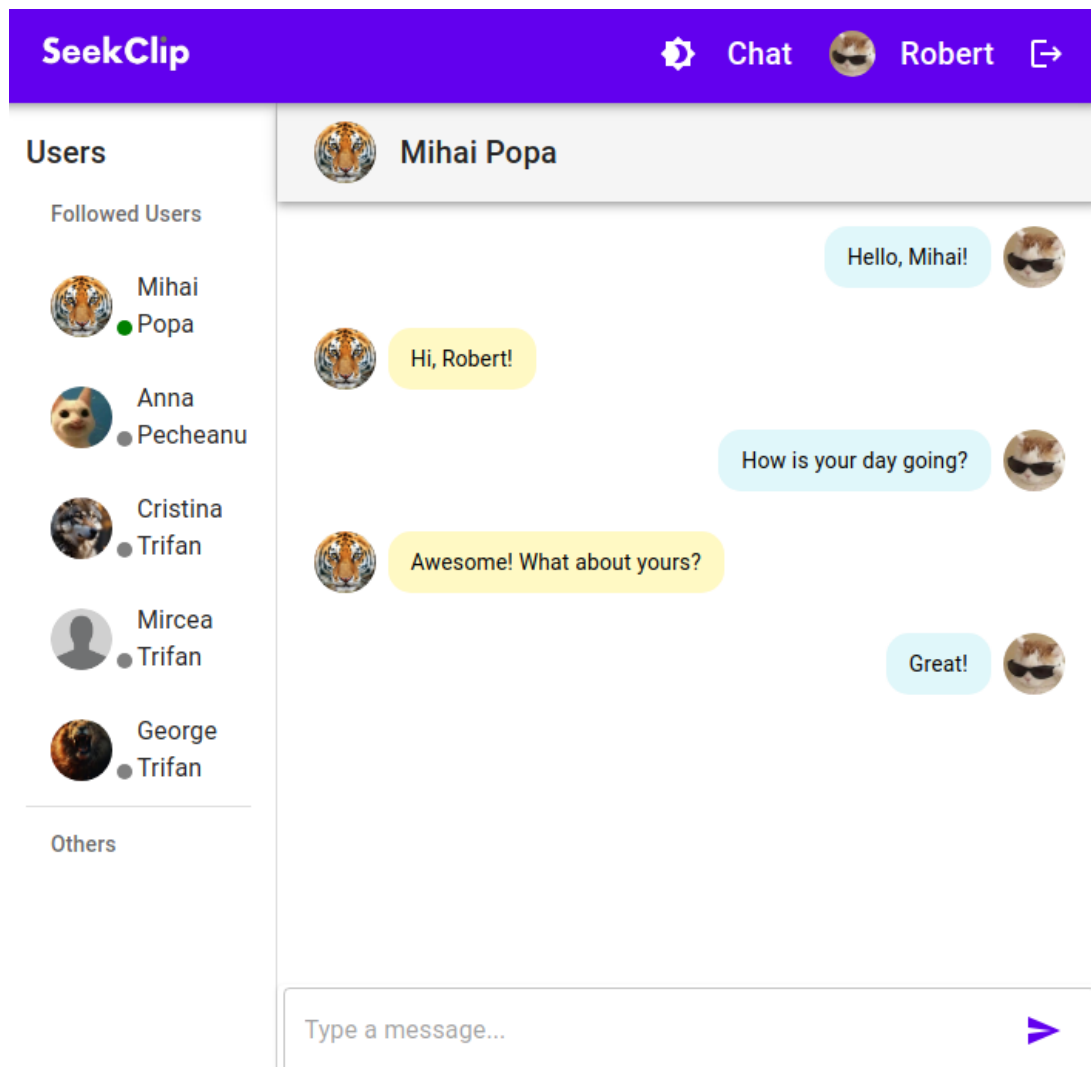


Figura 3.11: Pagina de conversație

Mai multe detalii despre serviciul de mesagerie și socket-uri se găsesc în secțiunea 3.3.2.

### 3.2.7 Temă

Întreaga aplicație web poate fi personalizată prin intermediul a două teme: lumină și întuneric. Utilizatorul poate schimba tema din bara de navigare cu ajutorul unui buton care schimbă valoarea unei variabile din contextul temei și propagă această schimbare în întreaga aplicație web. (Fig. 3.12, 3.13)

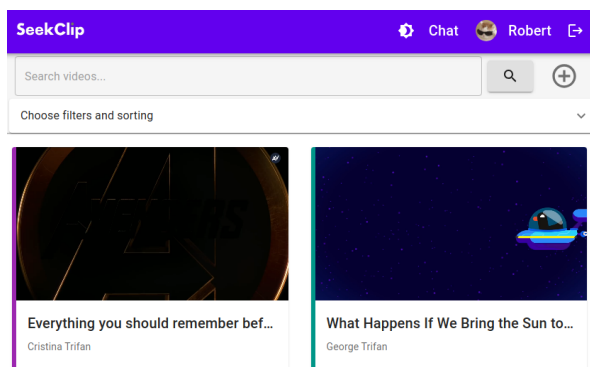


Figura 3.12: Tema lumină

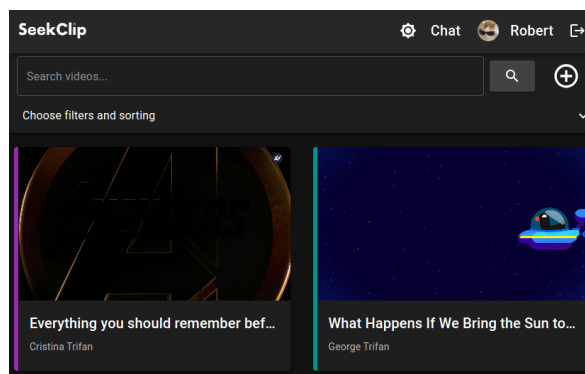


Figura 3.13: Tema întuneric

### 3.2.8 Rute protejate

Deoarece unele pagini impun existența token-ului JWT, implicit a utilizatorului autentificat, sau pot fi accesate doar de autor, am folosit un sistem de rute protejate. Acest sistem verifică accesul și redirecționează utilizatorul către alte pagini în cazul în care accesul nu este permis.

- **Rute protejate pentru utilizatorii autentificați:** se asigură că sunt accesate doar de utilizatori autentificați, în caz contrar fiind redirecționați către pagina de logare; de exemplu, pagina de profil, încărcarea videoclipurilor, postarea de comentarii/aprecieri.
- **Rute protejate pentru autorii videoclipurilor:** se asigură că sunt accesate doar de autorii videoclipurilor, și implicit de utilizatorii autentificați, în caz contrar fiind redirecționați către pagina principală; de exemplu, editarea videoclipurilor.

## 3.3 Backend

### 3.3.1 Express Server - Procesarea cererilor

Am creat un server Node.js folosind Express.js în limbajul de programare TypeScript care gestionează cererile primite de la client și le rutează către serviciile corespunzătoare.

Astfel, codul este structurat în:



- **Rute:** definesc rutele serverului și asociază cererile primite cu funcțiile corespunzătoare din controlere.
- **Controlere:** conțin funcțiile care procesează cererile primite de la client și apelează serviciile necesare.
- **Funcții intermediare:** funcții care se execută între primirea cererilor și procesarea acestora, adăugând/verificând informații suplimentare din cererea primită.
- **Modele:** definește structura entităților folosite în baza de date MongoDB și a interfețelor pentru a asigura tipurile de date.
- **Utilitare:** funcții utilitare care facilitează procesarea cererilor.
- **Configurări:** inițializează comunicarea cu alte servicii precum baza de date Elasticsearch.

## REST API

Comunicarea între client și server se face prin intermediul unui REST API. Prin definiție, REST (*Representational State Transfer*) impune existența a 3 componente: client, server și resurse.

Clientul îl reprezintă aplicația web care face cereri către server, serverul este aplicația care gestionează cererile și returnează răspunsuri, iar resursele sunt datele stocate în baza de date pe care serverul le manipulează.

Conceptul de REST API are la bază 6 principii:

- **Separarea client-server:** clientul și serverul sunt entități separate și izolate, serverul nu poate face cereri către client, doar clientul poate face cereri către server; în acest fel, cele două entități pot evolua independent.
- **Interfață Uniformă (*Uniform Interface*):** având în vedere diversitatea limbajelor de programare folosite atât pentru client cât și pentru server, este necesară o interfață comună, independentă de limbaj, care să faciliteze comunicarea între cele două entități; astfel REST API este construit peste protocolul HTTP și folosește metodele standardizate GET, POST, PUT, DELETE.
- **Cereri independente (*Stateless*):** fiecare cerere primită de la client conține toate informațiile necesare pentru a fi procesată cu succes, serverul nefiind nevoit să păstreze informații despre cererile anterioare.
- **Sistem multistrat (*Layered System*):** cererea primită de la client poate trece prin mai multe straturi intermediare folosite pentru a îmbunătăți performanța, securitatea sau scalabilitatea aplicației.

- **Depozitabil (*Cacheable*):** cererile anterioare pot fi stocate în *cache* pentru a fi refolosite la cereri ulterioare, reducând astfel timpul de răspuns și traficul de date între client și server.
- **Cod executabil (*Code on Demand*) (Optional):** serverul poate trimite cod executabil către client pentru a fi rulat în browser, însă această funcționalitate este opțională

## Codurile de stare HTTP

Pentru a comunica rezultatul unei cereri, există un standard de coduri HTTP împărțit în următoarele 5 categorii:

- **1xx:** cod returnat atunci când cererea a fost primită, dar procesarea nu a fost finalizată. De exemplu, când se face o îmbunătățire a protocolului de comunicare. (**informații**)
- **2xx:** cod returnat când cererea a fost gestionată cu succes. (**succes**)
- **3xx:** cod returnat atunci când clientul a fost redirecționat către o altă resursă. (**redirecționare**)
- **4xx:** erori provenite de la client, clientul a făcut o cerere greșită. (**client**)
- **5xx:** erori provenite de la server, serverul a întâmpinat o problemă în procesarea cererii. (**server**)

## Rute

Rutele sunt construite recursiv, plecând de la rădăcină și adăugând pentru fiecare API un nou nivel de rute. În cadrul aplicației noastre, am definit următoarele rute:

- **/api/auth:** rutele responsabile de autentificarea utilizatorilor, care mai apoi se specializează în /api/auth/register și /api/auth/login.
- **/api/user:** rutele responsabile de gestionarea informațiilor utilizatorilor și a interacțiunilor între aceștia, precum urmărirea/retragerea urmăririi unui utilizator.
- **/api/video:** rutele responsabile de gestionarea videoclipurilor, se specializează în operațiile CRUD (creare, citire, actualizare, ștergere) pentru videoclipuri și a interacțiunilor cu acestea.
- **/api/comment:** rutele responsabile de gestionarea comentariilor, aduc funcționalități precum adăugarea, editarea și ștergerea comentariilor.
- **/api/message:** rutele responsabile de gestionarea mesajelor, permit citirea acestora din baza de date. (trimiterea mesajelor este gestionată de WebSocket)

## Funcții intermediare

Funcțiile intermediare (*Middlewares*) sunt funcții care se execută între primirea cereri și procesarea acesteia, de obicei adăugând sau verificând informații suplimentare din cererea primită. În cadrul aplicației noastre, am folosit două funcții intermediare:

- **Autentificare:** se adaugă la cererile care necesită ca utilizatorul să fie autentificat, verificând astfel existența token-ului JWT și dacă acesta este valid. Verificarea se face prin decriptarea token-ului și verificarea semnăturii cu ajutorul secretului stocat pe server. Fiecare cerere are în header-ul ei un câmp *Bearer Token* care conține token-ul JWT. Odată decriptat, se adaugă în corpul cererii informațiile despre utilizatorul autentificat.
- **Multer:** se folosește pentru cererile care necesită încărcarea unor fișiere de tip imagine sau videoclip; se asigură că fișierul are formatul corect (de exemplu, imaginea este de tip *.jpg/.png/.jpeg* și videoclipul este de tip *.mp4/.mov/.avi*). Acesta creează un director în interiorul căruia se salvează fișierul încărcat și adaugă calea către fișier în baza de date. Serverul expune apoi url-ul către id-ul fișierului încărcat pentru a putea fi accesat de client.

## Modele

Modelele sunt folosite pentru a defini structura entităților folosite în baza de date MongoDB. Deci, pentru fiecare entitate din baza de date, definim un schelet (*Schema*) care conține câmpurile, tipurile de date ale acestora, restricțiile și referințele către alte entități. În cadrul aplicației noastre, am definit următoarele modele:

- **Utilizator (*User*):** reține pentru fiecare utilizator numele, prenumele, adresa de email, numele de utilizator, parola criptată, poza de profil și două liste cu persoanele care îl urmăresc și persoanele pe care le urmărește. De asemenea, adaugă și restricții pentru a asigura unicitatea numelui de utilizator și a adresei de email, tipuri de date pentru fiecare câmp, necesitatea completării câmpurilor și lungimea minimă și maximă a câmpurilor.
- **Video (*Video*):** reține pentru videoclip informații precum url-ul unde este stocat, titlul, descrierea, id-ul autorului, textul din videoclip, calea către subtitrări în formatul *.vtt*, o listă cu id-urile utilizatorilor care au apreciat sau nu postarea, topicul videoclipului și o listă de hashtag-uri. Restricțiile impuse sunt similare cu cele de la modelul anterior, au grijă ca fiecare videoclip să aibă obligatoriu metadatele necesare și să aibă un autor valid. De asemenea, sunt menționate și tipurile de date ale câmpurilor.

- **Comentariu (*Comment*)**: reține pentru fiecare comentariu textul, id-ul autorului, id-ul videoclipului, dacă este șters complet sau nu (*soft-delete/hard-delete*) pentru eficientizarea procesului de ștergere a comentariilor, și id-ul comentariului la care răspunde pentru a putea crea ierarhia comentariilor. Schema modelului impune restricția ca id-ul autorului și id-ul videoclipului să fie obligatorii, iar textul să fie completat.
- **Mesaj (*Message*)**: reține pentru fiecare mesaj conținutul său, id-ul autorului și id-ul destinatarului, precum și data la care a fost trimis.
- **Conversație (*Conversation*)**: reține pentru fiecare conversație o listă cu id-urile participanților la conversație precum și o listă cu id-urile mesajelor trimise în cadrul conversației.

## Controlere

Fiecare rută menționată anterior are asociată un controler care procesează cererea primită, apelează serviciile necesare și returnează răspunsul.

Menționez aici câteva dintre funcțiile mai importante:

- **register/login**: funcții care se ocupă de înregistrarea și logarea utilizatorilor, creează token-ul JWT folosind secretul stocat pe server și id-ul utilizatorului, verifică existența utilizatorului în baza de date și returnează token-ul JWT.
- **getUserByUsername/getUserId/updateUserProfile**: funcții folosite pentru a obține informații despre utilizator, respectiv pentru a actualiza informațiile utilizatorului. Se observă aici cum funcția de actualizare a profilului folosește atât funcția intermediară de autentificare pentru a se asigura că doar utilizatorul autentificat poate actualiza profilul, cât și funcția intermediară de încărcare a fișierelor pentru a actualiza poza de profil. Ambele funcții accesează baza de date pentru a obține sau actualiza informațiile și returnează răspunsul.
- **followUser/unfollowUser/isFollowing**: funcții care permit utilizatorului să urmărească sau să nu un alt utilizator și să verifice dacă urmărește un alt utilizator. În mod natural, funcția intermediară impune ca utilizatorul să fie autentificat pentru a putea executa aceste cereri.
- **getFollowers/getFollowing**: funcții care returnează lista cu utilizatorii urmăriți de un utilizator sau lista cu utilizatorii care îl urmăresc pe un utilizator.
- **addComment/updateComment/deleteComment/getCommentsForVideo**: după cum sugerează și numele, sunt funcții care permit adăugarea, actualizarea, ștergerea și obținerea comentariilor pentru un videoclip. Funcțiile de adăugare și

actualizare a comentariilor folosesc funcția intermediară de autentificare pentru a se asigura că doar utilizatorii autentificați pot executa aceste operații. Întâlnim aici și funcția *populate* din Mongoose care ne permite să expandăm referințele din baza de date, în cazul nostru, să obținem informații despre autorul comentariului.

- **uploadVideo/updateVideo/deleteVideo:** funcții care permit încărcarea videoclipurilor, actualizarea și ștergerea acestora. Fiecare funcție necesită funcția intermediară de autentificare pentru a asociera unui videoclip cu autorul său, iar funcțiile de actualizare și ștergere a videoclipurilor permit doar autorului videoclipului să execute aceste operații. Serviciul de încărcare a video-urilor apelează la rândul lui cele două servere de Flask pentru extragerea subtitrărilor și clasificarea videoclipului.
- **getAllVideos/getVideosByUser/getVideoById:** funcții care permit accesarea videoclipurilor din baza de date în diferite moduri: toate videoclipurile, videoclipurile unui utilizator sau un videoclip anume după id-ul său. Funcțiile caută la rândul lor în baza de date după criteriile specificate și returnează răspunsul.
- **likeVideo/dislikeVideo/getLikes/getDislikes/increaseViewCount:** funcții care permit interacțiunea cu un videoclip, adăugând sau ștergând id-ul utilizatorului care a dat apreciat sau nu postarea din lista de referințe și care incrementează numărul de vizualizări ale videoclipului la fiecare accesare. De asemenea, sunt tratate aici și cazurile speciale în care utilizatorul dă dislike după ce a dat like sau invers și cazul în care utilizatorul retrage aprecierea.
- **searchVideos:** funcție care primește din cerere cuvintele cheie (titlu, descriere, subtitrări, topic), accesează baza de date Elasticsearch pentru a căuta videoclipurile care conțin acele cuvinte și returnează o listă cu id-urile și scorurile videoclipurilor găsite. De menționat că am ales un prag (*threshold*) de 1.0 pentru a filtra rezultatele irelevante și am setat parametrul de confuzie (*fuzziness*) pe *AUTO* pentru a permite căutări aproximative.
- **getMessages:** funcție care primește din corpul cererii id-ul destinatarului și id-ul utilizatorului autentificat din funcția intermediară și accesează apoi baza de date pentru a returna sub formă de listă de referințe către mesaje, conversațiile dintre cei doi utilizatori.

## Utilitare

Funcțiile utilitare nu depind de o anumită entitate și sunt folosite pentru a facilita procesarea cererilor. În cadrul aplicației noastre, am definit următoarele funcții utilitare:

- **getTranscription:** înainte de a trimite cererea către serverul de Flask pentru recunoașterea vorbirii, e nevoie să extragem întâi secvența audio din videoclip

deoarece modelul lucrează cu fișiere audio. Folosesc, pentru acest lucru, librăria *ffmpeg-extract-audio* care permite procesarea fișierelor video și extragerea secvenței audio. De asemenea, secvența audio fiind destul de mare, creez un flux de date folosind *fs.createReadStream* și atașez acest flux la cererea către serverul de Flask.

- **getTopic:** după ce am obținut subtitrările videoclipului, folosesc un serviciu extern pentru a clasifica fiecare videoclip în funcție de metadatele sale și a subtitrărilor. Astfel, creez o cerere către serverul de Flask pentru clasificare și returnez topicul videoclipului.

## Configurări

Configurările sunt folosite pentru a inițializa comunicarea cu alte servicii precum baza de date Elasticsearch. Folosesc două fișiere de configurare: *elasticsearchClient.ts* și *elasticsearchSetup.ts*. Primul creează un client Elasticsearch folosind librăria *@elastic/elasticsearch* și exportă clientul pentru a fi folosit în servicii, iar al doilea creează un index specific Elasticsearch pentru a stoca informații necesare căutării videoclipurilor. Deoarece aplicația este un prototip, folosesc o singură replică pentru index și un singur shard.

## Expunerea datelor

Datele multimedia precum: videoclipurile, subtitrările și pozele de profil sunt stocate pe mașina serverului, în directoare separate: */uploads*, */captions* și */profile-pictures*. Aceste directoare sunt expuse către client folosind *express.static* și pot fi accesate la link-ul unde rulează serverul concatenat cu numele fișierului stocat în baza de date.

```
app.use('/uploads', express.static('uploads'));
app.use('/captions', express.static('captions'));
app.use('/profile-pictures', express.static('profile-pictures'));
```

## Criptarea parolelor

Serverul folosește algoritmul de criptare *bcrypt* pentru a stoca parolele utilizatorilor în baza de date. Algoritmul de criptare are la bază următoarele concepte:

- **Adăugarea valorilor aleatoare (*Salting*):** librăria *bcrypt* generează automat un o valoare aleatoare (*salt*) pentru fiecare parolă, pentru a preveni atacurile de tip *rainbow table*. Valoarea aleatoare este adăugată la parolă înainte de a fi criptată. Astfel, chiar dacă două parole sunt identice, valorile aleatoare diferite vor genera criptări diferite.
- **Factorul de cost (*Cost Factor*):** procesul de criptare implică aplicarea algoritmului de hashing de un anumit număr de ori, numit factorul de cost (*cost factor*). Acest



- **BACKEND\_TOPIC\_URL**: link-ul către serverul de Flask pentru clasificarea videoclipurilor.
- **BASE\_URL**: link-ul către serverul Express, folosit pentru a construi link-urile.
- **MONGO\_URI**: link-ul către baza de date MongoDB, conține și credențialele de acces.

## Pornirea serverului

Am configurat câmpul *scripts* din fișierul *package.json* pentru a customiza comenzile de pornire a serverului. Astfel, pentru modul de dezvoltare, am definit comanda *dev* care pornește serverul folosind *nodemon src/server.ts* și repornește serverul automat la modificarea fișierelor. Pentru modul de producție, am definit comanda *start* care pornește serverul folosind *node dist/server.js*.

Codul este structurat în două directoare */src* și */dist*, care codul sursă de codul compilat. Astfel, pentru eficientizarea codului în producție, codul este compilat o singură dată și rulat folosind fișierul *dist/server.js*. Pentru a face acest lucru, am folosit librăria *tsc* care dă *build* și rulează comanda *npx tsc*.

### 3.3.2 Socket.io Server - Comunicare în timp real

**Socket.io** este o librărie specifică Node.js, care are la bază *Engine.io* și oferă suport atât pentru metode de comunicare în timp real precum *WebSocket*, cât și pentru metode mai vechi precum *long-polling* deoarece, din motive de securitate, unele rețele blochează comunicarea *WebSocket*.

Socket.io facilitează comunicarea în timp real cu ajutorul conceptului de **canale bidirecționale** (*sockets*) care permit emiterea și ascultarea evenimentelor între client și server. Practic, serverul creează câte un proces pentru fiecare client, se ocupă de comunicarea între procese și de transmiterea evenimentelor între acestea.

De asemenea, librăria Socket.io introduce conceptul de **camere** (*rooms*) identificate unic printr-o adresă, care permite gruparea clienților în funcție de anumite criterii și izolează comunicarea cu un subset din toți clienții conectați.

În cadrul aplicației, am folosit Socket.io pentru serviciul de mesagerie și pentru utilizatorii activi folosind următoarele evenimente:

- **conectare** (*connection*): de fiecare dată când un utilizator se conectează la server, acesta emite evenimentul de conectare către server, serverul extrage id-ul utilizatorului din token-ul JWT și id-ul canalului bidirecțional și îl salvează într-un dicționar. De asemenea, la conectare, serverul verifică mesajele primite în absența utilizatorului și emite evenimente de tipul mesaj-nou (*new-message*) pentru fiecare mesaj primit.



- **deconectare (*disconnect*)**: acest eveniment este emis de utilizator când se deconectează de la server. În acest fel, putem actualiza lista utilizatorilor activi, putem șterge utilizatorul din dicționar și putem emite evenimentul de utilizatori-activi (*online-users*) care notifică toți utilizatorii activi despre schimbarea listei.
- **trimite-mesaj (*send-message*)**: eveniment emis de utilizator la fiecare mesaj trimis. Astfel, serverul joacă rolul de intermediar care primește mesajul, îl salvează în baza de date și emite evenimentul de mesaj-nou dacă destinatarul este activ.
- **mesaj-nou (*new-message*)**: eveniment emis de server către destinatarul mesajului când acesta este activ.
- **utilizatori-activi (*online-users*)**: eveniment emis de server către toate canalele deschise pentru a notifica schimbarea listei de utilizatori activi.
- **cerere-utilizatori-activi (*request-online-users*)**: eveniment emis de client când deschide pagina de conversații pentru a cere serverului lista utilizatorilor activi.

### 3.3.3 Flask Server - Recunoașterea vorbirii

Deși este un server de Flask, structura este similară cu cea a serverului Express în sensul că este împărțit în app, rute, controlere, config și utils.

#### Rute

Sintaxa specifică Flask-ului pentru definirea rutelor constă în a folosi decoratorul `@app.route` pentru a asocia o funcție cu o cale specifică și pentru a specifica metoda HTTP folosită.

- **/transcribe**: ruta care primește cererea de la serverul Express, extrage secvența audio, verifică ca extensia fișierului să fie validă, creează un fișier temporar pentru a stoca secvența audio, apelează funcția de recunoaștere a vorbirii și returnează textul transcris.
- **/vtt/<filename>**: ruta care returnează subtitrările în formatul .vtt pentru videoclip. Funcția primește numele fișierului și returnează subtitrările stocate în acel fișier.
- **/vtt/delete/<filename>**: ruta care șterge subtitrările pentru videoclip. Această funcție este apelată după ce subtitrările au fost livrate cu succes pentru a elibera spațiul de stocare.

#### Controlere

Pentru a menține logica din rute cât mai simplă, am creat un controler care se ocupă de procesarea secvențelor audio primite de la serverul Express.

## Procesarea secvențelor audio

Funcția **speech\_to\_text** (vorbire\_text) primește ca parametru calea către fișierul audio salvat temporar, îl deschide folosind *soundfile* și îl transformă într-un șir de float-uri. Sunt două tipuri de fișiere audio: **mono** și **stereo**, iar funcția trebuie să facă distincția între cele două tipuri.

Formatul *mono* are un singur canal audio, iar formatul *stereo* are două canale audio, fiind conceput pentru a reda sunetul într-un mod mai realist. Deoarece modelul de recunoaștere a vorbirii acceptă doar fișiere audio mono, am făcut media celor două canale pentru a obține un singur canal audio.

De asemenea, modelul a fost antrenat pe fișiere audio cu o frecvență de eșantionare de 16kHz, așa că în cazul în care frecvența fișierului audio este diferită (de exemplu, 44.1kHz), trebuie să o convertim la 16kHz. Acest lucru se face folosind funcția *resample* din librăria *resampy*.

Un aspect important, după cum am menționat și la seturile de date folosite, este că modelul a fost antrenat pe fișiere audio cu o durată de aproximativ 10 secunde. O secvență audio prea mare oricum nu ar încăpea în memoria serverului, așa că am ales să sparg secvența audio în segmente de câte 30 de secunde folosind funcția *stream* din librăria *librosa*.

Un alt aspect de luat în calcul este că, chiar dacă modelul ar putea procesa secvențe mai lungi, mai intervine și timeout-ul socket-ului care ar putea expira în cazul unor secvențe prea mari. Deși în cazul spargerii secvenței audio în segmente de câte 30 de secunde, se pot pierde informații de la începutul și finalul fiecărei bucăți, am ales să sacrific aceste informații pentru a asigura procesarea cu succes a secvențelor audio. Exceptând aceste cazuri, bucățile pot fi tratate independent și deci procesate în paralel, îmbunătățind semnificativ timpul de răspuns.

## Subtitrări

Funcția **make\_subtitles** (creare\_subtitrări) primește ca parametru textul transcris din fiecare bloc de 30 de secunde și creează subtitrările în formatul .vtt. În cadrul unui bloc, modelul de recunoaștere a vorbirii folosește următoarele etape:

- **Procesare:** modelul primește secvența audio și folosește procesorul specific de pe Hugging Face pentru a converti secvența audio în tensori. Vom numi acești tensori *input\_values*.
- **Predicție:** modelul primește *input\_values* și returnează *logits*, tot sub forma de tensori reprezentând probabilitățile pentru fiecare literă din vocabular. Sunt calculate apoi *predicted\_ids*, fiind literele cu cele mai mari probabilități.

- **Decodare:** folosind *logits*, modelul folosește algoritmul de decodare pentru a converti tensorii în text și obținem *transcription*.

Astfel, obținem pentru fiecare bloc *transcription*, *input\_values* și *predicted\_ids*.

Pentru a crea subtitrările, folosim următoarele etape:

- **ids\_w\_time:** cunoscând durata întregului bloc (30 de secunde) și numărul de token-uri prezis de model, putem estima momentul de timp al fiecărui token prin împărțirea indexului token-ului la numărul total de token-uri și înmulțirea cu durata blocului. Putem elimina token-urile pentru [PAD].
- **split\_ids\_w\_time:** folosind token-ul de delimitare între cuvinte, putem crea câte un grup de token-uri între fiecare delimitator. Practic, obținem cuvintele propriu-zise, dar acum știm și momentul de timp.
- **word\_timestamps:** pentru fiecare cuvânt, calculăm momentul de timp de început și de sfârșit ca fiind minimul, respectiv maximul dintre toate momentele de timp ale token-urilor din acel cuvânt.
- **group\_timestamps:** putem grupa acum cuvintele în blocuri de 7 cuvinte, pentru a contura subtitrările. Începutul și sfârșitul fiecărui bloc de cuvinte este dat de momentul de timp al primului și ultimului cuvânt din bloc.

```

predicted_ids: [6, 6, 0, 8, 0, 4, 4, 0, 17, 0, 5, 5, 0, 7, 0, 6,
                \_____/                \_____/
                  to                    meet
0, 0, 4, 4, 0, 0, 18, 0, 0, 7, 0, 0, 12, 12, 4, 4, 6, 0, 0, 0, 8,
                \_____/                \_____/
                  |                    |
                  was                  to
0, 4, 0, 0, 20, 0, 0, 10, 0, 0, 9, 0, 14, 0, 4, 4, 4, 5, 5, 7, 0,
                \_____/                \_____/
                  |                    |
                  find                  each
19, 11, 0, 4, 4, 0, 0, 8, 8, 0, 6, 11, 11, 0, 5, 13, 0, 4, 4, 4]
_____/                \_____/                |
                  |                    other

```

cuvinte: ['to', 'meet', 'was', 'to', 'find', 'each', 'other']

id-uri pentru litere și delimitator, dar fără pad

```
[6, 6, 8, 4, 4, 17, 5, 5, 7, 6, 4, 4, 18, 7, 12, 12, 4, 4, 6, 8,
```

```

\_____/      \_____/      \_____/      \_/_/
  to      |      meet      |      was      |      to
4, 20, 10, 9, 14, 4, 4, 4, 5, 5, 7, 19, 11, 4, 4, 8, 8, 6, 11, 11,
| \_____/      |      \_____/      |      \_____/

5, 13, 4, 4, 4]
_____/      |

```

```
split_ids_w_time
```

```

'to' -> [(0.54, 6), (0.56, 6), (0.64, 8)]
'meet' -> [(0.74, 17), (0.78, 5), (0.80, 5), (0.84, 7), (0.88, 6)]
'was' -> [(1.48, 18), (1.54, 7), (1.60, 12), (1.62, 12)]
'to' -> [(1.69, 6), (1.77, 8)]
'find' -> [(1.87, 20), (2.01, 10), (2.07, 9), (2.11, 14)]
'each' -> [(2.21, 5), (2.23, 5), (2.25, 7), (2.29, 19), (2.31, 11)]
'other' -> [(2.43, 8), (2.45, 8), (2.49, 6), (2.51, 11), (2.53, 11),
            (2.57, 5), (2.59, 13)]

```

```
word_timestamps
```

```

'to' -> 0.54 - 0.64
'meet' -> 0.74 - 0.88
'was' -> 1.48 - 1.62
'to' -> 1.69 - 1.77
'find' -> 1.87 - 2.11
'each' -> 2.21 - 2.31
'other' -> 2.43 - 2.59

```

```
group_timestamps
```

```
'to meet was to find each other' -> 0.54 - 2.59
```

## Formatarea subtitrărilor

Funcția **save\_subtitles** (salvează\_subtitrări) primește ca parametru subtitrările și creează un fișier *.vtt*, respectând formatul specificat. Fișierele în formatul *.vtt* încep cu antetul *WEBVTT*, iar pentru fiecare grup de subtitrări se specifică momentul de timp de început și de sfârșit, urmat de textul subtitrării.

Mai jos este un exemplu de subtitrare în formatul *.vtt*:

WEBVTT

00:00:04.162 --> 00:00:06.424

sometimes math and physics conspire in ways

00:00:06.524 --> 00:00:07.665

that just feel too good to be

00:00:07.745 --> 00:00:09.706

true let's play a strange sort of

Inspirat din issue-ul de pe GitHub: <https://github.com/huggingface/transformers/issues/11307>

## Config

Configurările sunt folosite pentru a specifica serverului cum să proceseze cererile primite.

- **ASR\_REPO**: numele modelului de recunoaștere a vorbirii folosit pentru a transcrie secvențele audio.
- **BOOSTED\_LM**: dacă este setat pe *true*, folosește varianta îmbunătățită cu un model de limbaj bazat pe n-grame pentru recunoașterea vorbirii.
- **SAMPLE\_RATE**: frecvența de eșantionare a fișierelor audio, setată pe 16kHz pentru a respecta cerințele modelului.
- **BLOCK\_SIZE**: lungimea blocurilor folosite în fluxul de date, setată pe 30 de secunde.

### 3.3.4 Flask Server - Clasificarea videoclipurilor

Serverul de Flask pentru clasificarea videoclipurilor este similar cu cel pentru recunoașterea vorbirii, dar are o structură mai simplă, nefiind necesare mai multe etape de procesare.

## Route

Astfel, serverul definește o singură rută */topic* care primește cereri de tipul *POST* și returnează topicul textului primit. Deoarece secvența de intrare a modelului **BERT** este limitată la 512 token-uri, am ales să împart textul în blocuri de câte 512 token-uri și să le trimit succesiv la model pentru a obține topicul.

Pentru a obține topicul final, am observat că informațiile din primul bloc de date (care conțin titlul și descrierea videoclipului) tind să fie mai relevante decât restul blocurilor (care conțin doar subtitrările). Astfel, am ales să folosesc doar primele 10 blocuri și să măresc importanța contribuției primului bloc la clasificare.

În final, serverul contorizează numărul de apariții ale fiecărui topic și returnează topicul majoritar.

## Configurări

Serverul creează un *pipeline* specific *Hugging Face* înainte de a primi cereri, pentru a reduce timpul de răspuns. Acest *pipeline* conține modelul antrenat pe setul de date *BBC News*. Mai multe detalii se regăsesc în secțiunea 2.2.

### 3.3.5 Flask Server - Întrebări și răspunsuri pentru videoclipuri

Serverul de Flask pentru întrebări și răspunsuri pentru videoclipuri folosește API-ul pus la dispoziție de **ChatGPT** pentru a genera răspunsuri la întrebările legate de conținutul videoclipurilor. Inițializarea serverului presupune crearea unui client din librăria *openai*.

## Rute

Serverul creează o singură rută */qa* care primește cereri de tipul *POST* a căror conținut este un obiect JSON cu două câmpuri: *question* și *subtitles*. Astfel, este apelată funcția */ask\_gpt* care la rândul ei apelează API-ul de la OpenAI folosind modelul *gpt-3.5-turbo* pentru a genera răspunsul la întrebarea primită.

## 3.4 Baze de date

Pentru stocarea datelor, am ales să folosesc două baze de date: **MongoDB**, datorită structurii sale flexibile care permite dezvoltarea rapidă a aplicațiilor și **Elasticsearch**, pentru sistemul de căutare rapid în metadatele videoclipurilor.

### 3.4.1 MongoDB

#### Arhitectura bazei de date

MongoDB este o bază de date de tip **NoSQL** care stochează datele sub formă de documente în formatul *BSON*. Documentele sunt grupate în colecții, iar colecțiile sunt apoi grupate în baze de date.

Intuitiv, un document este echivalentul unui rând dintr-o bază de date relațională, dar cu o structură similară unui obiect *BSON*. Acesta permite stocarea datelor fără a respecta

un model de date fix, adică pot fi adăugate câmpuri noi, chiar și alte documente, diferite de celelalte documente din aceeași colecție.

## Structura datelor

Fiecare document conține, pe lângă datele propriu-zise, câmpuri speciale precum:

- **\_\_id**: un *ObjectId* adăugat automat de MongoDB pentru a identifica unic documentul.
- **\_\_v**: un câmp special care indică versiunea documentului în cazul actualizării. Acest câmp a fost conceput pentru a preveni conflictele de actualizare în cazul cererilor concurente.
- **createdAt** și **updatedAt**: câmpuri speciale de tipul *ISODate* care indică momentul de creare și actualizare pentru fiecare document.

## Colecții

În cadrul aplicației noastre, am definit 3 colecții:

- **utilizatori** (*users*): conține informații despre utilizatori precum: numele și prenumele, adresa de email, numele de utilizator, parola criptată și url-ul către poza de profil.
- **videoclipuri** (*videos*): conține informații despre videoclipuri precum: titlul, descrierea, topicul, id-ul autorului (referință către colecția *users*), url-ul către videoclip, url-ul către subtitrări și două liste separate pentru id-urile utilizatorilor care au dat like și dislike.
- **comentarii** (*comments*): conține informații despre comentarii precum: textul comentariului, id-ul autorului (referință către colecția *users*), id-ul videoclipului (referință către colecția *videos*), *isDeleted* (folosit la *soft-delete* și *hard-delete*), *parentId* în cazul în care comentariul este un răspuns la alt comentariu.
- **mesaje** (*messages*): conține pentru fiecare mesaj: id-ul utilizatorului care a trimis mesajul, id-ul destinatarului, textul mesajului, momentul de timp al trimiterii și câmpul *isDelivered* folosit pentru obținerea mesajelor necitite.
- **conversație** (*conversation*): conține pentru fiecare conversație o listă cu id-urile participanților la conversație și o listă cu id-urile mesajelor trimise.

## Interacțiunea cu baza de date

Interacțiunea cu baza de date are loc cu ajutorul librăriei *mongoose* care definește modelele în serverul Express și execută operații de tip CRUD (Creează, Citește, Actualizează, Șterge) în baza de date. *mongoose* este un **ODM** (*Object Data Modeling*) care oferă un

nivel de abstractizare peste MongoDB și permite definirea de scheme pentru documente, validarea datelor și crearea de relații între colecții.

Un alt aspect important pus la dispoziție de **MongoDB** îl reprezintă capacitatea de a crea replici la nivel de colecție, numite *indexes*. Deși aplicația noastră este un prototip, nu are nevoie de scalările orizontale, dar în cazul unui volum mare de date, replicile facilitează operațiile de citire și scriere, îmbunătățind timpul de răspuns.

### 3.4.2 Elasticsearch

#### Concept

Elasticsearch este un motor de căutare ce organizează datele în documente în formatul *JSON* care sunt apoi grupate în indici. Indicii sunt echivalentul bazelor de date din MongoDB, iar documentele sunt echivalentul rândurilor din tabelele relaționale.

Concepte cheie în Elasticsearch:

- **Document:** unitatea de bază folosită de Elasticsearch în formatul *JSON* (considerat formatul standard pentru transferul de date)
- **Indici:** o colecție de documente cu trăsături similare formează un indice și reprezintă cel mai înalt nivel de organizare a datelor.
- **Index inversat (*Inverted Index*):** reprezintă mecanismul care stă la baza motoarelor de căutare și constă într-o structură de date (similară cu un *hashmap*) care mapează cuvintele (sau mai bine zis, termenii) la locațiile unde apar în documente. Astfel, fiecare document este împărțit în termeni individuali care sunt folosiți la căutare.

Figura 3.14 ilustrează conceptul *Inverted Index*. [6]

#### Inserare

În cazul aplicației noastre, am definit un singur indice *videos* care conține metadatele videoclipurilor: titlul, descrierea, topicul și subtitrările. De fiecare dată când un videoclip este încărcat în aplicație, inserarea se face atât în baza de date MongoDB, cât și în Elasticsearch.

#### Căutare

Interogarea în Elasticsearch permite specificarea criteriului de căutare precum: textul întreg *full-text*, confuzie *fuzzy*, prefix, expresii regulate etc. În cazul aplicației noastre, am ales să folosesc *fuzziness AUTO* pentru a trece cu vederea greșelile de scriere și să folosesc *match* pentru a căuta în câmpurile menționate anterior.

---

<sup>3</sup>Imagine preluată de pe site-ul knowi.io la adresa: <https://www.knowi.com/blog/what-is-elastic-search/>



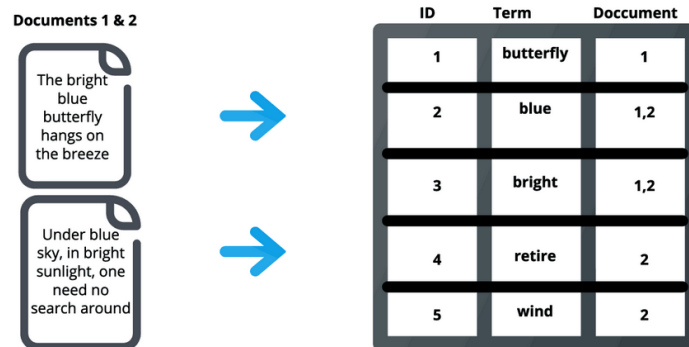


Figura 3.14: Inverted Index<sup>3</sup>

## 3.5 Servicii

### 3.5.1 Cron Jobs

#### Definiție și utilitate

Un **cron job** este un proces care rulează automat la momente de tip prestabilite. În cadrul proiectului, un *cron job* își găsește utilitatea în procesul de curățare al arborelui de comentarii pentru fiecare videoclip. Problema constă în găsirea unei soluții eficiente care șterge comentariile marcate drept *soft-delete* cu subarborele de răspunsuri gol.

Figura de mai jos ilustrează acest concept:

```
Cool video!
|__ Indeed it is!
|__ |__ [deleted] (1)
|__ [deleted] (2)
|__ |__ Waiting for the next one!
[deleted] (3)
Awesome!
|__ [deleted] (4)
|__ |__ Yeah, I agree!
|__ [deleted] (5)
|__ |__ [deleted] (6)
```

În exemplul de mai sus, comentariile cu textul [deleted] sunt marcate drept *soft-delete*, semnificând că deși au fost șterse de utilizator, ele încă există în baza de date.

Problema apare atunci când un comentariu este marcat drept *soft-delete*, dar nu știm, fără o parcurgere în prealabil, dacă poate fi șters sau nu.

O primă idee ar fi ca de fiecare dată când un comentariu este șters, să facem o cerere pentru întreg arborele de comentarii și să verificăm dacă fiecare nod are copii care pot fi la rândul lor șterși. Această metodă este inefficientă, deoarece necesită o cantitate mare

de resurse pentru a parcurge întreg arborele de comentarii la fiecare actualizare.

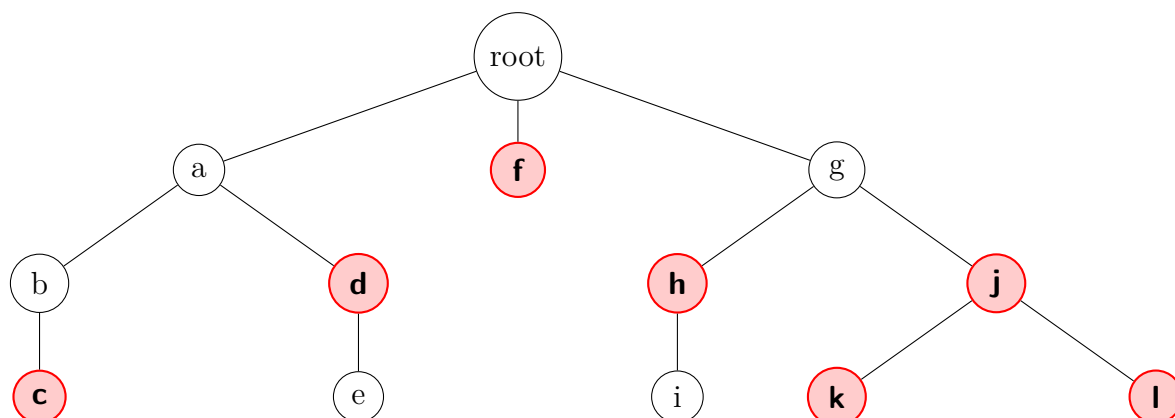
O altă soluție ar fi să păstrăm comentariile în starea *soft-delete* și să rulăm un *cron job* la intervale regulate de timp care să curețe arborele de comentarii.

## Algoritm

Algoritmul de curățare a arborelui de comentarii este următorul:

- **Inițializare:** creează conexiunea cu baza de date, obține toate comentariile și creează structura arborescentă menționată anterior.
- **Parcursere:** parcurge arborele în adâncime folosind algoritmul DFS (Depth First Search) și verifică recursiv folosind conceptul de programare dinamică, dacă un nod poate fi șters sau nu. La fiecare pas din recursie se verifică dacă nodul curent este marcat drept *soft-delete* și dacă toți subarborii copiilor săi sunt de asemenea marcați drept *soft-delete*. În caz afirmativ, nodul curent poate fi șters și este adăugat într-o listă de noduri *deletable*. Altfel, nodul nu poate fi șters și implicit toate nodurile de pe drumul până la rădăcină nu pot fi șterse.
- **Ștergere:** după ce arborele a fost parcurs și a fost creată lista *deletable*, se apelează iar clientul MongoDB pentru a șterge toate nodurile din listă.

Exemplul de mai jos ilustrează algoritmul de curățare a arborelui de comentarii:



În arborele prezentat, nodurile colorate cu roșu reprezintă comentariile *soft-delete*. Algoritmul va adăuga nodurile *c*, *f*, *j*, *k* și *l* în lista *deletable*.

Chiar dacă nodurile *d* și *h* sunt marcate drept *soft-delete*, ele nu pot fi șterse permanent (*hard-delete*) din cauza subarborilor activi.

## 3.6 Încărcare

### 3.6.1 Docker

#### Concept

**Docker** este o platformă care permite dezvoltatorilor să izoleze logica, dependențele și mediul de lucru în containere. Spre deosebire de mașinile virtuale, Docker nu virtualizează întregul sistem de operare, ci combină aplicația și dependențele sale cu librăriile oferite de sistemul de operare.

În contextul mașinilor virtuale (VM), alocarea de resurse este dirijată de un *hypervisor* care gestionează resursele fizice ale sistemului. În mod asemănător, Docker folosește un sistem de gestiune al resurselor mult mai eficient din punct de vedere al memoriei și timpului de rulare.

Avantajele folosirii containerelor Docker sunt:

- **memorie mică:** un container Docker nu conține întregul sistem de operare, ci doar procesele și dependențele necesare pentru aplicație, dimensiunea acestuia fiind de ordinul MB.
- **portabilitate:** rezolvă problema *it works on my machine* datorită izolării mediului de lucru cu ajutorul imaginilor și containerelor.
- **utilizare eficientă a resurselor:** permite gestionarea eficientă a resurselor sistemului.

#### Dockerfile și Imagini Docker

Imaginile Docker reprezintă unitatea de bază a containerelor și conțin tot ce este necesar (codul sursă, dependențele, librăriile, variabile de mediu etc.) pentru a rula o aplicație. Imaginile sunt structurate în nivele de abstractizare construite pe baza unui fișier numit *Dockerfile*.

Un *Dockerfile* enumeră instrucțiunile necesare pentru construirea imaginii. De obicei, se pleacă de la o imagine de bază (de exemplu, *python:3.8-slim*), după care se instalează dependențele necesare și se copiază codul sursă în container. În ultima parte a fișierului, se specifică comanda care pornește aplicația.

Odată ce *Dockerfile*-ul este definit, se pot construi imagini Docker folosind comanda *docker build*, eventual cu argumente suplimentare în linia de comandă. (de exemplu, menționarea portului pe care rulează aplicația, versiunea imaginii etc.)

#### Docker container

Un container de Docker folosește imaginea construită anterior și creează instanțe ale acesteia. În timp ce o imagine permite doar citirea, un container permite dezvoltatorului

interacțiunea cu acesta, fie pentru a rula comenzi, fie pentru a monitoriza starea aplicației.

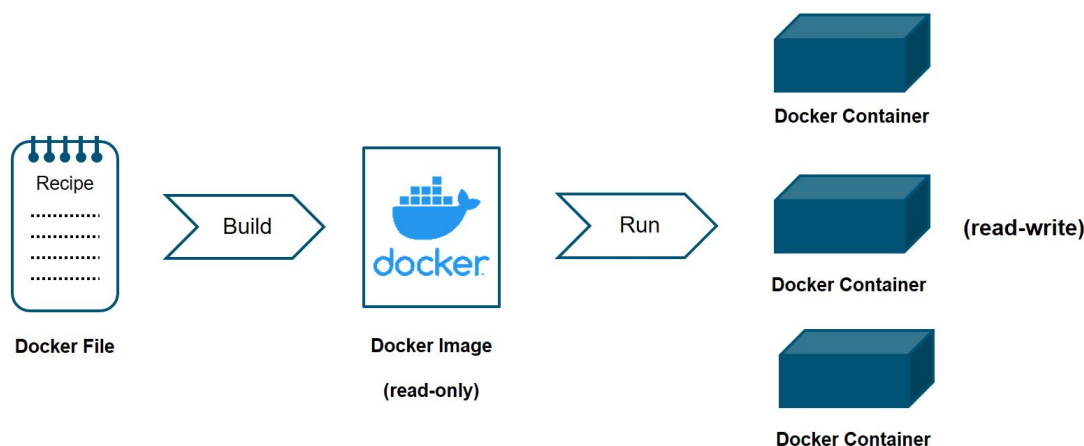


Figura 3.15: Docker<sup>4</sup>

## Docker Compose

Pe parcursul dezvoltării aplicației, apar din ce în ce mai multe servicii care trebuie gestionate separat și care depind unele de altele. *Docker Compose* este un tool care înglobează toate comenzile necesare pentru pornirea și oprirea serviciilor, precum și pentru construirea imaginilor și a containerelor.

*Docker Compose* folosește un fișier numit *docker-compose.yml* care conține toate detaliile necesare pentru fiecare serviciu, dintre care amintim: imaginea folosită, numele containerului, porturile expuse, variabilele de mediu, volumele, dependențele între alte servicii, rețelele folosite etc.

## Volume și rețele

În cazul în care aplicația are nevoie de o stocare persistentă, de pildă pentru bazele de date, se pot folosi volume Docker. Un volum Docker este, de fapt, un director din sistemul gazdă care este montat în container al cărui conținut este persistent chiar și după oprirea containerului.

De asemenea, Docker oferă și posibilitatea de a crea rețele virtuale prin intermediul căroră serviciile pot comunica între ele. Implicit, Docker oricum atribuia fiecărui container o adresă IP, dar în cazul pornirilor succesive ale aceluiași container, adresa IP se schimbă și nu mai poate fi accesată. Folosirea rețelelor Docker permite atribuirea manuală a IP-urilor.

---

<sup>4</sup>Imagine preluată de pe site-ul *SureSoft* la adresa: <https://suresoft.gitlab-pages.rz.tu-bs.de/workshop-website/continuous-integration/containers.html>

### 3.6.2 Dockerizarea aplicației

Aplicația beneficiază de avantajele oferite de *Docker* și *Docker Compose* pentru a simplifica procesul de dezvoltare și de încărcare. În cadrul proiectului, am definit următoarele servicii:

- **MongoDB Container:** rulează o instanță de MongoDB pentru stocarea datelor, folosind un volum Docker pentru a asigura persistența datelor și făcând o corespondență între portul gazdă și portul containerului.
- **Elasticsearch Container:** rulează o instanță de Elasticsearch pentru căutarea eficientă în metadatele videoclipurilor, folosind de asemenea un volum Docker.
- **Flask Server - Recunoașterea Vorbirii:** rulează serverul Flask pentru recunoașterea vorbirii, folosind portul 5001. Deoarece serverul comunică cu MongoDB și Elasticsearch, cele două servicii sunt specificate ca dependențe.
- **Flask Server - Clasificarea Videoclipurilor:** rulează serverul Flask pentru clasificarea videoclipurilor, folosind portul 5003. Nu are dependențe.
- **Flask Server - Întrebări și Răspunsuri:** rulează serverul Flask pentru întrebări și răspunsuri, folosind portul 5004. Nu are dependențe.

Pentru serverele de *React* și *Node.js* am ales să nu folosesc containere *Docker* astfel încât să pot beneficia de o dezvoltare rapidă și de actualizare în timp real a codului sursă. La nivel de producție însă, cele două servere pot fi containerizate și ele.

# Capitolul 4

## Concluzie

### 4.1 Concluzie

Prin realizarea acestei lucrări am prezentat conceptul de recunoaștere a vorbirii dintr-un videoclip integrându-l într-o aplicație web relevantă. Deși scopul proiectului a fost o aplicație web, librăria Material UI oferă posibilitatea de a fi folosită și pe dispozitivele mobile.

Inițial, mi-am îndreptat atenția către modelul de recunoaștere a vorbirii și generarea subtitrărilor, dar consider că integrarea acestuia într-un context concret ilustrează mai bine utilitatea și importanța acestuia. De aceea, mi-am structurat lucrarea de licență în două părți: partea de **Concepte teoretice despre învățare automată**, în care am detaliat arhitectura modelului *wav2vec 2.0*, cum am ales setul de date și cum am antrenat modelul, și arhitectura modelului *BERT* pentru clasificarea videoclipurilor, explicată în aceeași manieră, folosită pentru a îmbunătăți algoritmul de căutare, precum și partea de **Prezentare a aplicației**, în care am prezentat arhitectura aplicației web, tehnologiile folosite, structura frontend-ului, backend-ului, bazelor de date, dar și a serviciilor de încărcare și a serviciilor utilitare (Cron Jobs).

Așa cum am menționat în secțiunea *Prezentarea aplicației*, aplicația oferă utilizatorilor funcționalități precum: autentificare, CRUD pe informațiile proprii, dar și pe videoclipuri și interacțiunea cu acestea, căutarea videoclipurilor după cuvinte cheie și suport pentru recunoașterea vorbirii, crearea subtitrărilor și clasificarea videoclipurilor.

În procesul de dezvoltare al aplicației, am gândit întâi funcționalitățile pe care le urmăresc și apoi am ales tehnologiile potrivite pentru a le implementa. Acest context mi-a oferit oportunitatea de a învăța aceste tehnologii și de a le aprofunda în lucrarea de licență.

Am optat pentru stack-ul **MERN** (MongoDB, Express.js, React.js, Node.js), împreună cu limbajul TypeScript, la care am adăugat și trei servere de Flask în Python. Am ales să fac partea de antrenare a modelelor tot în Python datorită multitudinii de librării

disponibile.

Personal, m-am confruntat cu lipsa subtitrărilor în filme și tutoriale și consider că această problemă nu este încă rezolvată. Cu cât domeniul de recunoaștere a vorbirii evoluează mai mult și apar seturi de date pentru mai multe limbi, cu atât mai multe persoane vor beneficia de o experiență mai personalizată în înțelegerea conținutului video. De asemenea, consider că procesul manual prin care omul adaugă subtitrări este unul consumator de timp și resurse, iar o soluție precum cea prezentată în această lucrare poate simplifica munca depusă.

În plus, consider că integrarea API-ului de la ChatGPT pentru răspunderea la întrebări legate de conținutul videoclipurilor este o funcționalitate cu mult potențial, care poate îmbunătăți procesul de căutare și înțelegere a conținutului video.

În concluzie, lucrarea de licență prezintă un prototip de aplicație web care integrează recunoașterea vorbirii și generarea subtitrărilor în contextul videoclipurilor și oferă utilizatorilor o experiență mai bună în înțelegerea conținutului video.

## 4.2 Perspective

În timpul dezvoltării am observat câteva aspecte care pot fi îmbunătățite atât pentru o performanță mai bună a aplicației, cât și pentru o utilitate mai extinsă. Printre acestea se numără:

- **Suprimarea zgomotului:** am observat că unele videoclipuri au melodii sau zgomote de fundal care afectează performanța modelului de recunoaștere a vorbirii. Suprimarea acestor sunete ar putea îmbunătăți calitatea subtitrărilor.
- **Traducerea subtitrărilor:** m-am concentrat pe limba engleză, având mai multe resurse disponibile, dar consider că modelul de recunoaștere a vorbirii poate fi folosit și pentru alte limbi (fiind și țelul la care țintim cu această aplicație), fie prin antrenarea unui model separat, fie prin traducerea subtitrărilor generate.
- **Seturi de date mai mari:** am folosit seturi de date mici (*MiniLibriSpeech*, *Common Voice Delta Segment 16.1*) din lipsa resurselor computaționale de antrenare, dar consider că antrenarea pe seturi de date mai mari ar îmbunătăți performanța modelului.
- **Topicuri mai diverse:** videoclipurile sunt clasificate în 5 categorii: (tehnologie, sport, afaceri, politică, divertisment), dar natura materialelor video este mult mai diversă de atât. O clasificare mai detaliată ar îmbunătăți experiența utilizatorilor.
- **Corector de greșeli ortografice:** deși am îmbunătățit modelul de recunoaștere a vorbirii cu un model de limbaj bazat pe n-gramme, tot mai apar cuvinte greșite.

Un corector de greșeli ortografice (*spell checker*) ar putea îmbunătăți calitatea subtitrărilor.

- **Suport pentru subtitrări:** librăria *transformers* de pe Huggingface nu oferă suport pentru generarea subtitrărilor și a trebuit să prelucrez separat momentele de timp ale cuvintelor și să le sincronizez cu videoclipul. În implementarea mea am ales să grupez câte 7 cuvinte într-o subtitrare, dar în mod normal, această valoare ar trebui să fie ajustată la viteza de vorbire.
- **Procesarea paralelă a secvențelor audio:** după cum am menționat mai sus, fiecare secvență audio este împărțită în bucăți de 30 de secunde, iar aceste bucăți sunt procesate secvențial. Nefiind dependente între ele, pot fi procesate în paralel.
- **Echilibrarea sarcinii de lucru (*Load Balancing*):** fiind un prototip, am folosit un singur server pentru toate serviciile, dar într-un mediu de producție, arhitecturi precum *nginx* sau *Kubernetes* sunt vitale pentru existența aplicației.
- **Replici pentru bazele de date:** în aceeași manieră ca la punctul anterior, am folosit o singură bază de date *MongoDB* și *Elasticsearch*, dar în producție, arhitecturi de tip *Master-Slave* sau *Sharding* îmbunătățesc viteza de răspuns.
- **Autentificare sigură:** am folosit autentificare cu *token JWT*, dar în producție, servicii de autentificare precum *Auth0* sau *Firebase* sunt mai sigure.

Și lista poate continua. Procesul de dezvoltare adoptat a fost unul concentrat pe finalizarea funcționalităților promise, dar intrând în detalii, sunt multe aspecte care pot fi îmbunătățite.



# Bibliografie

- [1] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers și Gregor Weber, *Common Voice: A Massively-Multilingual Speech Corpus*, 2020, arXiv: [1912.06670 \[cs.CL\]](#).
- [2] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed și Michael Auli, *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*, 2020, arXiv: [2006.11477 \[cs.CL\]](#).
- [3] Pamela Bump, „How Video Consumption is Changing in 2023”, în *HubSpot* (Apr. 2023), URL: <https://blog.hubspot.com/marketing/how-video-consumption-is-changing>.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee și Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019, arXiv: [1810.04805 \[cs.CL\]](#).
- [5] Elasticsearch, URL: <https://www.elastic.co/>.
- [6] Jay Gopalakrishnan, „Elasticsearch: What it is, How it works, and what it’s used for”, URL: <https://www.knowi.com/blog/what-is-elastic-search/>.
- [7] Derek Greene și Pádraig Cunningham, „Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering”, *Proc. 23rd International Conference on Machine learning (ICML’06)*, ACM Press, 2006, pp. 377–384.
- [8] Kenneth Heafield, „KenLM: Faster and Smaller Language Model Queries”, *Proceedings of the Sixth Workshop on Statistical Machine Translation*, ed. de Chris Callison-Burch, Philipp Koehn, Christof Monz și Omar F. Zaidan, Edinburgh, Scotland: Association for Computational Linguistics, Iul. 2011, pp. 187–197, URL: <https://aclanthology.org/W11-2123>.
- [9] Eric Jang, Shixiang Gu și Ben Poole, *Categorical Reparameterization with Gumbel-Softmax*, 2017, arXiv: [1611.01144 \[stat.ML\]](#).
- [10] Vassil Panayotov, Guoguo Chen, Daniel Povey și Sanjeev Khudanpur, „Librispeech: An ASR corpus based on public domain audio books” (2015), pp. 5206–5210, DOI: [10.1109/ICASSP.2015.7178964](#).

- [11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai și Soumith Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019, arXiv: [1912.01703 \[cs.LG\]](#).
- [12] Steffen Schneider, Alexei Baevski, Ronan Collobert și Michael Auli, *wav2vec: Unsupervised Pre-training for Speech Recognition*, 2019, arXiv: [1904.05862 \[cs.CL\]](#).
- [13] Jörg Tiedemann, „Parallel Data, Tools and Interfaces in OPUS”, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, ed. de Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk și Stelios Piperidis, Istanbul, Turkey: European Language Resources Association (ELRA), Mai 2012, pp. 2214–2218, URL: [http://www.lrec-conf.org/proceedings/lrec2012/pdf/463\\_Paper.pdf](http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf).
- [14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser și Illia Polosukhin, *Attention Is All You Need*, 2023, arXiv: [1706.03762 \[cs.CL\]](#).