***DU MELLÉKLET 6.1.***

PÉCSI TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

MATEMATIKAI ÉS INFORMATIKAI INTÉZET

**Webes alapú videómegosztó platform fejlesztése**



Témavezető: Dr. TÉMAVEZETŐ NEVE

témavezető beosztása

Külső konzulens: Dr. TÉMA VEZETŐ

Készítette: TUSNÁDI BÁLINT (K3Z7XE)

PROGRAMTERVEZŐ

INFORMATIKUS SZAK

PÉCS, 2018

***DU MELLÉKLET 6.2.***

*“Nem a legerősebb faj lesz a túlélő, nem is a legintelligensebb, hanem az, amelyik a leggyorsabban képes változni.”*

(Darwin, 1859)

1. Bevezetés

A közösségi média napjainkban az internet jelentős részét teszi ki, és

általánosságban elmondható, hogy az én generációm, az úgynevezett „Z

generáció” mindennapjainak szerves része. A közösségi média egy részét a

hosszabb formátumú, tartalmasabb videók alkotják, azonban ezen a területen

jelentős monopóliumot vélek felfedezni, ami ösztönzött arra, hogy elkészítsem

ezt a projektet.

A szakdolgozatomban egy nyilvános, webes alapú videó- és egyéb

tartalommegosztó platform fejlesztési folyamatát mutatom be. A fejlesztés

során különböző keretrendszereket és könyvtárakat fogok használni,

amelyeket részletesen bemutatok, és használatukat indokolom. Emellett a

fejlesztés módszertanai is bemutatásra kerülnek.

A szoftvernek képesnek kell lennie webböngésző segítségével videók

böngészésére, felhasználók regisztrálására és bejelentkezésére, valamint a

felhasználóknak lehetőséget kell biztosítania videók feltöltésére és

megjegyzések nyilvánossá tételére a videók alatt. Továbbá, ha a szakdolgozat

tartalma megköveteli, a szoftvernek lehetővé kell tennie a felhasználók

számára önálló megjegyzések és élő közvetítések közzétételét is. Mivel a

szoftver internetes alapú lesz, a biztonság kiemelten fontos szerepet kap (pl.

SQL injection, egyszerű jelszavak elleni védelem, IDOR (Insecure Direct Object Reference)).

A szakdolgozat kliens-szerver architechtúrára épül. Állni fog egy szerver (backend) oldalból, mely a kiszolgáló szerver számítógépen fog lefutni és egy kliens (frontend) oldalból mely az ügyfél számítógépén fut le. Továbbá állni fog egy adatbázisból, amely természetesen a szerver oldalon helyezkedik majd el.

Ahogy feljebb is említettem, igyekszem minél több keretrendszert és könyvtárat felhasználni, azomban érdemes megemlíteni, hogy miért is használunk egyártalán keretrendszereket vagy könyvtárakat egyártalán. Felgyorsítják a szoftverfejlesztés folyamatát, ugyanis ha már valaki megvalósította az adott funkciót, akkor minek valósítsuk meg úrja? Egy szoftverfejlesztő egyik legfontosabb tulajdonsága, hogy hajlandó legyen megismerkedni új technológiákkal, ugyanis ez a terület igen gyorsan halad és előfordulhat hogy éppen egy olyan technológia vagy programrészlet jelenik meg, mely felgyorsíthatja a szoftverfejlesztést.

Ezek után felmerülhet a kérdés, hogy mi a könyvtár és mi a keretrendszer és mi a különbség. Mindkettő lényegében egy kódrészlet. A legnagyobb különbség közöttük az úgynevezett „Inversion of control”, azaz kinél van az irányítás? A fejlesztő vagy a keretrendszer? Amikor egy könyvtárat használunk, akkor mi döntjük el, hogy mikor hívjuk meg az adott függvényt, mikor hozunk létre egy könyvárban definiált típusú osztályt (Pl. egy stdio printf). A keretrendszerek esetén azonban nem mi, a fejlesztők döntjük el, hanem előre medvannak adva, hol kell meghívni az adott függvényeket. Pl. a PHP Laravel-ben a projekt létrejöttekor létrejön egy „Routes” mappa, ahova kell tenni a webapi kéréseket.

A szoftver elengedhetetlen része egy webapi fejlesztése, melyet Spring boot-ban tervezek megalkotni. A Spring Boot egy modern, Java nyelven alapuló keretrendszer, amely leegyszerűsíti a projekt backend fejlesztését számos moduljának köszönhetően. Emellett biztonsága is kiváló, és a Java nyelv természetéből fakadóan a kód egyszeri megírása után bármilyen platformon futtatható. A ezen döntésemet hosszas mérlegelés után hoztam meg. Figyelembe vettem a keretrendszer népszerűségét és egyszerűségét is. Habár ez a kettő kéz a kézben jár, ugyanis nem lenne a világon az egyik legnépszerűbb keretrendszere, ha nem lenne viszonylag könnyű használni.

Továbbá figyelembe vettem a teljesítményt is amely a mai hardverek esetén is fontos növekvő projektek esetén. Ez a statisztika a 2024 leghasználtabb keretrendszereit mutatja meg, melyet a statista.com weboldalán lehet megtalálni. Ez a statisztika a laikus ember számára megtévesztő lehet, ugyanis a Spring Boot csak 12,9%-on áll. Ez a statisztika MINDEN webes keretrendszert bele számít, nem csak a szerveroldalit. A React, jQuery, Next.js, Angular, Vue.js mind leginkább vagy kifejezetten csak kliens oldali keretrendszer. És ezek után marad a Node.js, Express, ASP.NET CORE, ASP.NET, Flask és a Spring Boot. A Node.js egy JavaScript futtató környezet, azaz képes lefuttatni JavaScript kódot böngésző nélkül akár szerver oldalon. Ez lehetővé tette rengeteg JavaScript-ben írt szerveroldali webes keretrendszer létrejöttét, példálul az Express-t. Az Express vagy Express.js vagy Node.js Express egy szerver oldali Webes keretrendszer melynek a programozási nyelve a JavaScipt és igen egyszerű használni. Az Express egy valóban jó választás lehetett volna a Spring Boot helyett is. Azonban személyes tapasztalatom alapján úgy éreztem, hogy a szoftverfejlesztő cégeket sokkal inább érdekli a Spring Boot mintsem Node.js Express. Továbbá a medium.com kísérletei szerint a Node.js Express 3-szor lassabb, mint a Spring Boot. Ezen kívül még azt is megemlítik, hogy a Spring Boot több processzort és memóriát használ, ami arra enged következtetni engem, hogy sokkal jobban kihasználja a számítógépet, mint a Node.js Express. Az ASP.NET egy .NET szerver oldali keretrendszer, melyet a Microsoft fejlesztett. A programozási nyelve a C#. A Flask meg egy szerver oldali Python keretrendszer. Ezekkel a keretrendszerekkel nincs tapasztalatom, ezért sem válaszottam ezeket. Továbbá érdemes megemlíteni a PHP-t és ehhez kapcsolódó keretrendszereket, mint pédálul a Laravel. 20 évvel ezelőtt talán a PHP volt a legnépszerűbb szerver oldali webes technológia és még mai napig is gyakran használják már létező webes szoftverek karban tartására. Azonban már elavultá vált a alacsony hatékonysága és szintaxisa miatt. Még kismillió választás van mint példálul a Rust nyelven íródott keretrendszerek. A Rust egy programozási nyelv mely a legtöbb modern programozási nyelvvel ellentétben nem használ úgynevezett „Garbage collector”-t, mely memória tisztítására szolgál a memória szivárgás elkerülése érdekében. A Rust egy igen forradalmi módszert használ egy úgynevezett „system of ownership” módszert a memória szivárgás elkerüléséhez. Ez a módszer azt takarja, hogy különböző szabályok alapján a fordító program megbizonyosodik, hogy ki az adott memóriarész tulajdonosa. Így nem kell futási időben egy „Garbage collector”-nak futnia így felgyorsul a rendszer. Ez miatt a fejlesztők körében a legimádottabb programozási nyelvvé vált a Stackoverflow kérdőív alapján.

A projekt egy másik aspektusa a kliens (frontend) oldal. Ezt le lehetne akár fejleszteni egy alap HTML-ben is azonban a feljebb említett indokok miatt érdemesebb erre is egy könyvtárat választani. A React-et választottam, mivel ez az egyik legnépszerűbb és a komponens alapú architechtúrája miatt az adott komponensek könnyen újra felhasználhatóak lesznek. Emellett még sok más előnye van mint példálul, hogy széleskörűen lehet használni és igen könnyű reszponzív dizájnt csinálni.

Azonban a Spring Boot közvetlen nincs a React integrálva. Ez azt jelenti, hogy külön kell létrehoznunk egy Spring Boot és egy React projektet. Mindkettő lényegében egy-egy önnálló szerverként működne. A React szolgálná ki az UI-t és kérné le a létező adatokat vagy küldené el az új adatokat a Spring Boot REST API-nak. A szerveroldal azonosítaná a felhasználó jogosultságát, feldolgozná a kérést és elmentetné az adatot egy fájlrendszerbe vagy adatbázisba.

Az adatbázist megemlítve, érdemes elgondolkozni az adatok tárolásának módszeréről. Ehhez át kell tekinteni, mit is tervezünk eltárolni. Mivel a szakdolgozat egy videómegosztó platform készítéséről szól, ezért nyilvánvalóan el kell tárolnunk videókat, melyek akár egyenként gigabájtos nagyságot is elérhetik. Másrészt el tervezünk tárolni felhasználó adatokat is mint pédálul felhasználónév, jelszó, email stb. amelyek minössze néhány bájtos mennyiségű adatok egyenként. Ezekhez a bájtos nagyságú adatokat érdemes eltárolni egy SQL adatbázisba az egyszerű és gyors és egyszerű CRUD (Create, Read, Update, Delete) operációk miatt.

Az SQL-adatbázisok, más néven relációs adatbázisok olyan rendszerek, amelyek táblagyűjteményeket tárolnak, és táblázatos, oszlopok és sorok formájában rendszerezik a strukturált adathalmazokat, mint a számolótáblák.

Azonban videókat nem érdemes SQL-ben tárolni ugyanis a műveletek nagy mennyiségű adatokon jelentősen lelassíthatják a query-ket (lekérdezéseket). Ez helyett érdemesebb eltárolni egy fájlrendszerbe és valami egyedi azonosítóval ellátni, melyet már berakhatjuk az SQL adatbázisba.

Rengeteg féle SQL van, én a MariaDB mellett döntöttem. A MariaDB egy nyílt forráskódú SQL-alapú relációsadatbázis-kezelő szerver, egy MySQL-fork. Gyors, rugalmas és ezzel az SQL adatbázissal van a legtöbb tapasztalatom.

A kliens oldalra visszatérve érdemes lesz egy CSS keretrendszert is választani. Ezekből is rengeteg választás van de csak kétféle között érdemes választani, Bootstrap és Tailwind. Mindkettő lényegében kész css osztályokat szolgálatat, melyeket hozzárendelhetünk HTML elemekhez. A Bootstrap egyszerűbb és lényegében kész sablonokat szolgáltat. Azonban Tailwind sokkal személyre szabhatóbb, optimalizáltabb és könnyű sima CSS-el kombinálni, egy külön CSS fájlba is írhatunk tailwind osztályokat, melyekkel csökkenthetjük a kód redundanciát és könnyebb a dizájnt egységesíteni. Ezen okok miatt döntöttem a Tailwind mellett, de nem kell a Bootstrap-ről sem lemondani. A fejlesztés eleén érdemes Bootstrap-et használni, hogy könnyen lehessen egy alap webdizájnt létrehozni, hogy jobban látszódjon a projekt összképe, de ahogy nő a projekt érdemes átváltani Tailwind-re.

A kód íráshoz természetesen kell egy program. Technikailag elég egy jegyzettömb és egy cmd is de erősen ajánlott egy IDE használata. Az integrált fejlesztői környezetek vagy röviden IDE-k (Integrated Development Environment) alkalmazások, melyek jelentősen megkönnyítik és meggyorsítják a szoftverfejlesztést. Én a Visual Studio Code-ot (továbbiakban: VSCode) fogom használni. Ez az IDE egyszerű, ingyenes, nyílt forráskódú, könnyen személyre szabható és sokkal gyorsabb mint más alternatívák pl. Jetbrains szoftverek.

Programtervezési minták: Programozásban általános problémák megoldásához nyújt mintát. Felgyorsítja és megkönnyíti egy applikáció fejlesztését, mert a fejlesztőnek már csak bele kell implementálnia a logikát a mintába.

MVC minta: Model-View-Controller (magyarul: Modell-Nézet-Vezérlő) egy tervezési minta, mely szétszedi az applikációt három fő komponensre:

Model: Az adatot és az üzleti logikát írja le. Gyakran objektumok tulajdonságait írja le egy ORM segítségével.

ORM: Az ORM Object-Relational Mapping (magyarul: objektum-relációs leképzés) egy programozási technika, ami összekapcsolja a OOP programozási nyelveket relációs adatbázisokkal.

View: A felhasználói felületet írja le. Ez a kliens oldal.

Controller: Ez irányítja a kéréseket a modell és a nézet között.

Adatbázis architechtúrája:

Mit kell tárolnunk? El kell tárolnunk a felhasználókat, videó meta adatokat, kommenteket.

A táblák nevei:

users

video\_metadatas

comments

Felhasználó:

id: integer, unique, not null, primary key

Ez a felhasználó egyedi azonosítója.

username: varchar(255), not null

Ez a felhasználó neve, ezt a felhasználó maga választhatja kötelezően

email: varchar(255), unique, not null

Ez a felhasználó email címe. Az email nyilvánvalóan mindig egyedi, ezért nyugodtan adhatunk egy kényszert, hogy egyedi legyen. Kötelezővé tesszük, hogy majd tudjuk a felhasználót értesíteni és hogy fel tudjuk használni email-es megerősítésre.

password: varchar (255), not null

Ez a felhasználó jelszava, majd a webapi a bcrypt algoritmust felhasználva majd titkosítja. Ne tároljunk jelszavakat titkosítás nélkül!

created\_at: timestamp, not null, default: current\_timestamp

Ez a felhasználó létrehozásának dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

updated\_at: timestamp, not null, default: current\_timestamp, on update: current\_timestamp

Ez a felhasználó frissítésének dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

profile\_image\_filename (opcionális): char(36), not null, unique

Ez lesz a profilképek fájlneve. A videófájlokat UUID egyedi azonosító formátumban el tervezzük tárolni, amit a webapi réteg ad. Az UUID szabvány mindig pont 36 karakter hosszú.

followers (opcionális): integer, not null

Ez lesz a követők száma.

views (opcionális): integer, not null

Ez lesz az adott felhasználó összes videójának összesített nézettsége. Ennélkül le kéne kérdezni az összes videót leszűrni az adott felhasználóra és összeadni az összes nézettséget, mely nagy mennyiségű videó táblában hosszú ideig is eltarthat. Ezért, ha eltároljuk az adott videó nézettségét, akkor tároljuk el az adott felhasználóhoz is.

Videó:

id: integer, unique, not null, primary key

Ez a videó egyedi azonosítója.

user\_id: integer, not null

Ez a videó külső kulcsa, amit hozzárendelünk a felhasználó elsődleges kulcsához.

filename: char(36), not null, unique

Ez lesz a videók fájlneve. A videófájlokat UUID egyedi azonosító formátumban el tervezzük tárolni, amit a webapi réteg ad. Az UUID szabvány mindig pont 36 karakter hosszú.

title: varchar(255), not null

Ez a videó címe. A felhasználónak kötelező megadnia.

Description (opcionális): varchar(1000)

Ez a videó leírása. Max 1000 karakter. Nem lesz kötelező a felhasználónak megadnia.

views: integer, not null, default: 0

Ez a videó nézettsége.

likes (opcionális): integer, not null, default: 0

Ez a videó „lájk” száma.

dislikes (opcionális): integer, not null, default: 0

Ez a videó „diszlájk” száma.

status: tinyint

Ez a videó elérhetőségének státusza. Egybájtos egész szám az értéke. Lehetne akár szöveg is, de úgy gondolom, hogy a videó tábla nagysága az adatbázisban könnyen tud hízni, ezért adtam neki csak egy bájtot. A webapi-ban majd megfeleltethetük rendes szövegként a számokat.

0 = nyilvános, 1 = privát, 2 = rejtett. És még 253 másik értéket hozzáadhatunk, ha kell.

created\_at: timestamp, not null, default: current\_timestamp

Ez a videó létrehozásának dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

updated\_at: timestamp, not null, default: current\_timestamp, on update: current\_timestamp

Ez a videó frissítésének dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

Komment:

id: integer, unique, not null, primary key

Ez a komment egyedi azonosítója.

video\_id: integer, not null

Ez a komment egyik külső kulcsa. Egy videó alatt lehet több komment is.

user\_id: integer, not null

Ez a komment másik külső kulcsa. Kommenteket felhasználók töltenek fel. Ezért ez is kell.

created\_at: timestamp, not null, default: current\_timestamp

Ez a komment létrehozásának dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

updated\_at: timestamp, not null, default: current\_timestamp, on update: current\_timestamp

Ez a komment frissítésének dátuma és ideje. Ha elfelejtenénk, akkor az adatbázis ezt automatikusan kitölti. Arra figyeljünk, hogy a „timestamp” típus az UTC időzónában tárolja el.

Spring Boot projekt telepítése:

1. VSCode letöltése:

A code.visualstudio.com oldalán keresztül tölthetjük le, végigmegyünk a szokásos telepítés folyamatán és kész.

2. Java telepítése:

A Spring Boot Java nyelven alapul, ezért mielőtt telepíthetjük a Spring Boot-ot, fel kell telepítenünk a Java-t. Pontossabban a JDK-t (Java Development Kit), amely majd lehetővé teszi, hogy lefuttathassuk a Java kódunkat. A JDK-t a www.oracle.com/java/technologies/downloads/ weboldalon találhatjuk meg. Akármelyik verziót letölthetjük, a „Java archive” fülön kereszül régebbi verziókat is letölthetünk. Javasolt a legújabb verziót használni. Én a JDK 23 verziót fogom használni. Windows-ra telepítés esetén másoljuk ki, hogy hova telepítjük és a telepítjük és telepítés után nyissuk meg a környezeti vátozókat, nyissuk meg a rendszer változók alatt a „Path”-et és másoljuk be egy új sorba a kimásolt útvonalat és írjuk a sor végére, hogy „/bin”. Ez azért kell, hogy akárhol elérhessük egy terminálban. Nyissunk meg egy terminált és írjuk be, hogy „java –version”, hogy megbizonyosodjunk, hogy le sikerült tölteni.

3. Spring Boot telepítése:

Töltsük le a Spring Boot és java bővítményeket VSCode-ban. VSCode-on belül nyomjuk le a CTRL + SHIFT + X gombkombinációt, hogy elnavigáljunk a bővítmények fülre. Itt töltsük le Microsoft és a VMware által készített Spring Boot nevű és a Java nevű Microsoft és Oracle által készített. bővítményeket Következő lépés a projekt létrehozása.

Opció 1: Nyissuk meg a start.spring.io weboldalt.

Project: Maven. A Maven egy építéskezelő eszköz.

Language: Java

Spring Boot (Verzió): A legutóbbi stabil (nem snapshot).

Project Metadata: Ezek tetszés szerint. Erősen ajánlott csak angol karaktereket használni. Pl. nem „körte”, hanem „korte”.

Packaging: Jar (kötegelő formátum, ZIP-en alapul)

Java: Ugyanazt, mint amilyen JDK verzió van.

És a végén GENERATE gombra kell kattintani. Le fog tölteni egy .ZIP fájl. Ezt csomagoljuk ki és nyissuk meg VSCode-ban. Meg kell, hogy jelenjen egy futtatás gomb, ha rendesen etöltöttük a bővítményeket és mellette egy legördülő nyílnál meg kell, hogy jelenjen, hogy „Run Java”.

Opció 2: VSCode-on belül, nyomjuk le a CTRL + SHIFT + P kombinációt, írjuk be, hogy „Spring Initializr” és innentől ugyanaz, mint az 1. opció.

“Hello world!” projekt:

Amikor megismerkedünk egy programozási nyelvvel vagy technológiával, általában az első program amit írunk, az egy „Hello world!” projekt. Ezzel letesztelhetjük, hogy rendesen feltelepítettük-e a projektet. Csináljunk egy .java fált az src/main/com/<Group Id (“.” helyett “/” jel)>/<Artifact Id> relatÍv útvonalra (ott, ahol van egy “Application.jav”a végződésű fájl) és nevezzük el „HelloController.java”-nak. Másoljuk be a következő kódot:

package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController; @RestController

// Osztály és a fájl nevének ugyanannak kell lennie

public class HelloController {

@GetMapping("/") // útvonal

public String index() {

return "Hello world!";

}

}

A VSCode-ban meg kell, hogy jelenjen egy “lejátszás gomb”. Futtassuk le a projektet. Néhány másodperc várakozás után nyissunk meg egy böngészőt és írjuk be a „localhost:8080” címet. Meg kell hogy jelenjen, hogy „Hello world!”.

MySQL telepítése:

Menjünk fel a apachefriends.org oldalra és töltsük le a XAMPP-ot. Miután letöltöttük, telepítsük fel és indítsuk el az „Apache Web Server”-t és a „MySQL Database”-t. Miután elindítottuk, navigáljunk el a „localhost” címre és meg kell hogy jelenjen egy XAMPP weboldal. Emellett a fejlécen meg kell, hogy jelenjen egy „phpMyAdmin” gomb is.

A XAMPP egy ingyenes és könnyen használható apache disztribúció. Tartalmazza a MariaDB-t, PHP-t és Perl-t (Lásd a program nevét). Nekünk csak a MariaDB és a phpMyAdmin részére lesz majd szükségünk. A phpMyAdmin egy PHP-ban írt webes eszköz, ami segít MySQL és MariaDB adminisztrálásában.

Authentikáció:

Az authentikáció (hitelesítés) az internetes applikációk egy alapköve. Elengedhetetlen a kényes adatok biztonságos tárolásához. Ugyanúgy, ahogy kell egy kulcs a lakásba vagy házba belépéshez, kell egy jelszó az applikációban tárolt privát adainkoz való hozzáféréshez.

A mi applikációnban is elengedhetetlen, hogy legyen egy hitelesítés, a privát videók tárolásához és hogy megnehezítsük az esetleges támadók dolgát, akik rossz célra használnák fel az adott hardver erőforrást, amelyen a webes applikációnk futna.

Korábban, az adatbázis architectúrája részben már említettem, hogy ne tároljunk jelszavakat titkosítás nélkül. Erről majd a webapi fog gondoskodni. Szerencsére a Spring Security függőség megoldja ezt nekünk. Ez a függőség a bcrypt hash függvényt fogja használni.

Hash függvény: A kriptográfiai hash-függvény egy speciális típusú hash függvény, amelyet különféle kriptográfiai alkalmazásokban való használatra terveztek, beleértve a digitális aláírásokat, üzenet-hitelesítési kódokat és a hitelesítés egyéb formáit. Ezek a függvények döntő szerepet játszanak a modern információbiztonsági gyakorlatokban, különösen az olyan protokollokban, mint az SSL/TLS.

Bcrypt: A bcrypt egy jelszavakhoz szánt hash-függvény, melyet 1999-ben készítettek.

Registrációs program:

Most, hogy már telepítettünk egy Spring Boot-ot és egy MariaDB-t, tudunk csinálni egy regisztrációs programot. Ehhez fel fogjuk használni ezt a GitHub projektet: github.com/Alanlands1/springbootBackend. Ezt telepítsük fel. Utána navigáljunk el a pom.xml fájlhoz és adjuk hozzá ezt:

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

Ezzel hozzáadjuk a mysql függőséget. Továbbá ajánlott hozzáadni ezt a függőséget is:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

Ez a függőség számos segítséget szolgáltat, mint példálul a projekt automatikus újraindítását, ami felgyorsíthatja a visszajelzést, ha bármit módosítunk a projekten.

Ezután indítsuk el az Apache-t és a MySQL-t , utána lépjünk be a phpMyAdmin-ba, navigáljunk a „Databases” fülre és hozzunk létre egy új adatbázist „registration” néven. Továbbá a “src/main/resources/application.yml” fájlt nyissuk meg és másoljuk be a következőt egy az egyben:

server:

error:

include-message: always

include-binding-errors: always

spring:

datasource:

url: jdbc:mysql://localhost:3306/registration?useSSL=false&serverTimezone=UTC

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

hibernate:

ddl-auto: update

properties:

hibernate:

dialect: org.hibernate.dialect.MySQL8Dialect

format\_sql: true

show-sql: true

Most már el tudjuk indítani a projektet. Ugyanúgy a kész oldal a „localhost:8080” címen érhető el.

Videó lejátszás:

A webes applikációnk célja, hogy videót tudjunk lejátszani. Ez úgy működik, hogy az api bizonyos bufferekben (szeletekben) küldi a biteket amit a kliens oldal folyamatosan betölt és lejátsza. Ezt élesben láthatjuk a webes videólejátszó applikációkban egy pédálul szürke csíkként a folyamatjelző sávon.

Videólejátszó oldal:

Következő lépésként dobjunk hozzá egy videó közvetítő oldalt. Töltsük le a következő github projektet: github.com/saravanastar/video-streaming. Bontsuk ki és nyissuk meg egy fájlkezelőben. Másoljuk be a .java fájlokat a már létező regisztrációs applikációnk java fájlaihoz. Példálul a “VideoController.java” fájlt a „src/main/java/com/example/demo/Controller/” mappába. A bemásolt .java fájlok importálási útvonalait írjuk át úgy, ahogy a mi projektünk útvonalai néznek ki. Pl. a „package com.ask.home.videostream.controller;” sort a “VideoController.java” írjuk át „package com.example.demo.Controller;”-re. A “video” mappát a „src/main/resources/” útvonalon másoljuk át a mi projektünk azonos útvonalába.

Ezt a kódot adjuk hozzá az „application.yml”fájlhoz:

video:

content:

path: ${VIDEO\_CONTENT\_PATH:target/classes/video}

A „LocalFileContentAdapter.java” fájlban a 107 sorban a következő kódban: @Override

public Long getContentSize(ContentRequest contentRequest) {

return Optional.ofNullable(contentRequest).map(\_ -> Paths.get(getFilePath(localFilePath, contentRequest.getFilePath(), contentRequest.getFileName()))).map(this::sizeFromFile).orElse(0L);

}

Ebben a kódban a map() függvényben át kell nevezni az “\_” argunentmot tetszőlegesen pl. „request” névre.

Végül, importáljuk be a „pom.xml”-be a következő függőségeket:

<dependency>

<groupId>io.projectreactor</groupId>

<artifactId>reactor-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-webflux</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

Ezek után már elindíthatjuk a MySQL és Spring Boot szerverünket és letesztelhetjük, hogy jól csináltuk-e a következő címen: „localhost:8080/api/v1/videos/stream/mp4/toystory”

Egy videónak le kellene, hogy játszódjon.

Ha a lejátszás közben megnyitjuk a fejlesztői eszközt az F12 gombbal és rá megyünk a „Hálózat” vagy „Network” felületre, akkor láthatjuk, hogy kapunk rengeteg ~314KB méretű válaszokat. Ezek a videó bufferek jönnek egyesével. Ennek a méretét át tudjuk állítani a „ApplicationConstants.java” fájlban a következő részben bájtban értve:

public static final int CHUNK\_SIZE = 314700;

Azonban észrevehetjük, hogy a videót csak akkor lehet megnyitni, ha be vagyunk jelentkezve. Ezt a “SecurityConfig.java”-ban lehet módosítani. Példálul:

.authorizeHttpRequests(registry ->{

// Itt lehet beállítani, hogy milyen útvonalakat engedjen át bejelentkezés nélkül

registry.requestMatchers("/req/signup","/css/\*\*","/js/\*\*","/api/v1/videos/\*\*").permitAll();

registry.anyRequest().authenticated();

}).build();

Ebben a példában átengedjünk az összes "/api/v1/videos/stream/"-mel kezdődő elérési útvonalakat.

Fájl feltöltése:

A projektünk egy fontos eleme lesz, hogy a felhasználók ne csak nézni tudjanak videókat, hanem feltölteni is. Ehhez fel fogom használni a következő GitHub projektet: github.com/callicoder/spring-boot-file-upload-download-rest-api-example

Ezt a projektet töltsük le és ugyanúgy, ahogy az előző projektekben, itt is másoljuk át a .java, .html, .js és .css fájlokat a megfelelő helyükre. Továbbá a Github projektből másoljuk át az „application.properties” fájlt és írjuk át a “file.upload-dir=” változó értékét egy relatúv útvonalra, ahova szeretnénk a projektünkön belül tárolni a pl. videókat. Én példálul itt tárolom:

file.upload-dir=./src/main/resources/video

Amint készen vagyunk, indítsuk el a MySQL-t és a Spring Boot projektünket és navigáljunk el a „localhost:8080/” útvonlara. Itt tudjuk feltölteni a fáljunkat akár belehúzással. Próbáljuk ki egy .mp4 videóval. Töltsük fel és navigáljunk el a „localhost:8080/api/v1/videos/stream/mp4/<Fájltő neve>” útvonalra és le kellene, hogy játszódjon a videónknak.

Fájl validáció:

Jelenleg ez a fájl feltöltő funkció akármilyen fájlt befogad, természetesen az általunk megadott fájlméret alatt.

# Max file size.

spring.servlet.multipart.max-file-size=200MB

# Max Request Size

spring.servlet.multipart.max-request-size=215MB

Ez azonban nem ideális egy videó platform esetében. A fájlfeltöltést mindig validálni kell egyrészt, hogy fel tudja dolgozni az apink és le tudja játszani a felhasználónak. Másrészt biztonsági oka is van. Ha nem validáljuk rendesen a feltöltött fájlt, akkor előfordulhat, hogy egy esetleges támadó futtatható kódot rejt bele, így meghackelve a weboldalunkat. Továbbá a validácót végző könyvtárunkat / keretrendszerünket rendszeresen naprakészen kellene tartani.

Példálul 2025. április 14-én a 4chan nevezetű weboldal hasonlóan hackelték meg. A PDF feltöltő kódrészletük nem validálta, hogy a feltöltött fájluk valóban PDF-e. Ez alapján a támadók fel tudtak tölteni egy fájlt, ami a weboldal egy része lefuttatott.

A projektben én az Apache Tika eszköztárat fogom használni. Az Apache Tika egy eszköztár, amely azonosítja és kinyeri a metaadatot és a szöveget különböző fájlokból (példálul az MP4-ből). Ezt az eszköztárat rendszeresen naprakészen tartják és viszonylag könnyű beimportálni a Spring Boot projektbe.

A következő módon kell módosítanunk a programunkat, hogy működjön:

A „pom.xml” fájlba illeszzük be ezeket a könyvtárakat:

<dependency>

<groupId>org.apache.tika</groupId>

<artifactId>tika-core</artifactId>

<version>2.9.0</version>

</dependency>

<dependency>

<groupId>org.apache.tika</groupId>

<artifactId>tika-parsers-standard-package</artifactId>

<version>2.9.0</version>

</dependency>

Ezután a „FileStorageService.java” fájlban a

public String storeFile(MultipartFile file)

függvényt módosítsuk a következő módon:

public String storeFile(MultipartFile file) {

String fileName = StringUtils.cleanPath(file.getOriginalFilename());

Tika tika = new Tika();

String detectedType;

try {

detectedType = tika.detect(file.getInputStream());

} catch (IOException e) {

throw new FileStorageException("Could not read file to detect type.", e);

}

if (!detectedType.equals("video/mp4")) {

throw new FileStorageException("Invalid file type. Only actual MP4 videos are allowed.");

}

try {

// Sanity check

if (fileName.contains("..")) {

throw new FileStorageException("Sorry! Filename contains invalid path sequence " + fileName);

}

Path targetLocation = this.fileStorageLocation.resolve(fileName);

Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE\_EXISTING);

return fileName;

} catch (IOException ex) {

throw new FileStorageException("Could not store file " + fileName + ". Please try again!", ex);

}

}

Ne felejtsük el beimportálni a fájlba a könyvtárakat, ha esetleg az adott IDE nem tette volna meg. Ha kipróbáljuk, akkor láthatjuk, hogy csak MP4 fájlokat enged feltölteni. Ha esetleg nem működne a feltöltés, akkor mensük el a változtatásokat, zárjuk be és nyissuk meg újra az IDE-t.

Jelenleg a projektünk remekül fel tudja tölteni az MP4 fájlokat azonban nem tudunk még hozzá társítani lényeges adatokat, mint pédálul cím, leírás és így tovább…

Ehhez a fent írtak alapján létre kell hoznunk egy táblát a videónak. A tábla létrehozását automatikusan megoldhatjuk egy model és egy repository létrehozásával az apinkban. Ezután módosítsuk a „FileController.java” fájlban az „uploadFile()” függvényt, hogy:

- A kérésben befogadja, a cím és a leírás paramétereket

- A fájl név egy generált UUID legyen

- Csináljunk egy VideoModel pédányt és állítsuk be az elemeket megfelelően és mentsük el

- A „storeFile()” függvényt is módosítanunk kell, hogy befogadja az UUID fájlnevet

Csináljunk a „FileStorageService.java” fájlban egy új „storeFile()” függvényt a string fájlnév paraméterrel (Akár le is cserélhetjük az eredeti függvényt, de a Java metódus vagy függvény néven kívül paraméterek alapján is meg tud különböztetni). Továbbá nyilván a fájlnév változó deklarálást is kivehetjük a módosított függvényünkből.

Ezek után ha megnyitjuk a böngészőben, akkor láthatjuk, hogy nem tudjuk elmenteni a videót, hibát jelez. Ez azért van, mert a frontend nem küldi el a paramétereket. Helyette Postman-ben kell letesztelnünk. Nyissuk meg a Postman-t, válasszuk ki a POST kérést, a Headers-be be kell írnunk a sütinket, hogy authentikáljunk. Ezt a böngészőböl, fejleszői módban láthatjuk, ha küldünk egy kérést és megnézzük a „Request Headers”-t. Továbbá a „Body”-ban meg kell adnunk a cím, leírás és a fájl kulcsokat értékekkel. A fájl kulcsnál válasszuk ki a „File”-t. Ha elküldük, akkor láthatjuk, hogy megkaptuk a megfelelő választ. Ha átirányít a bejelentkezés felületre, akkor lehet, hogy a kulcsunk helytelen vagy lejárt.

Most, hogy fel tudjuk tölteni nem csak a videót, hanem az adatbázisban is be tudjuk benteni az adatait, vissza is kellene küldeni a felhasználónak. Ehhez először át kell tekinteni, hogy hogyan is fog működni. A kliens oldal először lekéri a metaadatokat és a fájl paraméterből külön egy példálul „fetch api”-val lekéri a videót pufferekben.

Először írjunk egy GET kérést, ami elküldi a video metaadatokat egy JSON formátumban példálul a „VideoController.java” fájlba. Ezután írjunk egy metódust, ami lekéri az adott rekordot ID alapján a „VideoRepository”-tól. Ezt a rekordot vissza is adhatjuk válaszként. Ezt le tudjuk tesztelni a böngészőben is és Postman-ben is. Sütikre jelenleg nincs szükség, ha azt az útvonalt adtuk meg, amikor definiáltuk a GET kérést, amit már kivételeztünk a „SecurityConfig.java”-ban.