Міністерство освіти та науки України
Запорізький інститут економіки та інформаційних технологій
Кафедра ІТ

ПОЯСНЮВАЛЬНА ЗАПИСКА
до курсової роботи

з дисципліни: «РОЗРОБКА ВЕБДОДАТКІВ ЗА ДОПОМОГОЮ ФРЕЙМВОРКІВ»

на тему: Створення веб-додатка для керування медіаколекцією з використанням Angular та можливість підтримки різних типів медіафайлів

Виконав:
Студент групи ІПЗ-111К9 _____ Пєлєван А.Є.

Науковий керівник:
Доцент _____ Мержинський Є.К.

Запоріжжя 2024

ЗАВДАННЯ:

1. Тема роботи: Створення веб-додатка для керування медіаколекцією з використанням Angular та можливість підтримки різних типів медіафайлів, затверджена рішенням кафедри (протокол № _____)

2. Термін подання студентом закінченої роботи на кафедру: 15.12.2024

3. Вихідні дані до роботи (проекту) (визначаються кількісні або (та) якісні показники, яким повинен відповідати об’єкт розробки): розробка веб-додатка для керування медіаколекцією з використанням Angular

4. Вимоги до змісту (перелік питань, що їх належить розробити) (визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту роботи) :

4.1 Розгляд предметної області та огляд існуючих аналогів;

4.2 Вибір та обґрунтування програмних засобів та технологій для реалізації проекту;

4.3 Реалізація проекту з використанням описаних методик та організація архітектури программи;

4.4 Налаштування та тестування створеного програмного продукту.

5. Індивідуальний план виконання

№ етапу	Зміст	Термін виконання
1	Формування теми курсового проекту	10.10.2024
2	Збір практичного матеріалу за темою	30.11.2024
3	Робота над створенням курсового проекту	10.12.2024
4	Оформлення пояснювальної записки та докладу	15.12.2024
5	Захист курсового проекту	24.12.2024

Керівник (ПІБ, підпис) _____

З планом ознайомлений (ПІБ, підпис) _____

РЕФЕРАТ

Пояснювальна записка складається з: 31 с., 9 рисунків, 3 джерел. Об’єкт дослідження – Веб-додаток для керування медіаколекцією з використанням Angular.

Мета роботи – Розробка веб-додатку для керування медіаколекцією у середовищі Visual Studio Code мовами TS, JS, HTML, CSS із застосуванням технології фреймворка Angular та бази даних Mongo DB.

Методи дослідження – використання технологій фреймворків, зберігання даних у базі даних, організація пошуку, завантаження та обробки даних, Visual Studio Code для розробки веб-додатків.

Проводиться аналіз засобів керування даними. Обґрунтовується вибір технології та середовища розробки веб-додатку. Розроблено веб-додаток для керування медіаколекцією, що забезпечує збереження, пошук та видалення медіафайлів.

ЗМІСТ

ВСТУП

Медіа-колекція є важливою складовою для зберігання медіафайлів. У цій роботі буде розглядатись процес розробки такого веб-додатка за допомогою фреймворку Angular у середовищі розробки Visual Studio. Основою системи є використання технологій Angular та Mongo DB для забезпечення ефективного зберігання, пошуку та керування файлами.

Метою даної курсової роботи є розробка теоретичних основ і практична реалізація медіа-колекції. Буде проведено огляд сучасних інструментів та бібліотек для створення такого веб-додатку, а також аналіз оптимальних методів їх використання. Результатом роботи стане веб-додаток, який забезпечує надійне та зручне керування медіафайлами.

Для повноцінної програми потрібно розробити кілька ключових компонентів: основні класи для зберігання інформації, методи для додавання, видалення, відтворення та пошуку, а також механізми для зберігання даних у БД і їх завантаження при запуску веб-додатку. Також важливо налаштувати взаємодію між різними частинами додатку.

У ході розробки будуть вивчені та реалізовані наступні кроки:

- Обдумати та створити архітектуру програми.
- Розробити моделі або схеми об'єктів програми.
- Реалізувати збереження файлів у БД.
- Налаштувати взаємодію між різними компонентами програми.
- Оглянути програму та зробити висновки.

Таким чином, дана робота сприятиме глибшому розумінню процесу розробки систем для керування даними з використанням сучасних технологій фреймворку Angular та середовища Visual Studio.

РОЗДІЛ 1 ОГЛЯД БАЗ ДАНИХ

1.1 Історія виникнення баз даних

Питання зберігання та обробки інформації завжди були важливими для людства. З розвитком технологій люди винаходили нові способи зберігання інформації, такі як поштові системи, архіви, бібліотеки тощо. З появою інформаційних технологій можливості стали ще більш глобальними. Одним із важливих кроків у розвитку збереження даних стало поява баз даних на комп’ютерах.

Інтерпол був заснований у 1923 році для координації міжнародних зусильу боротьбі зі злочинністю. Основою роботи Інтерполу є картотека злочинців, яка зберігає інформацію про міжнародних правопорушників та допомагає правоохоронним органам різних країн обмінюватися цією інформацією. Спочатку дані зберігалися в паперовій формі, що було незручно і неефективно.

З розвитком комп'ютерних технологій у другій половині XX століття картотека Інтерполу також почала використовувати комп'ютери для зберігання та обробки даних. Спочатку комп'ютери були великими та дорогими, і їх використовували лише для найважливіших завдань. Однак з часом, із розвитком комп'ютерних мереж та Інтернету, стало можливим об'єднати інформаційні ресурси різних країн і створити глобальну базу даних злочинців.

1960-ті рр. розроблення перших БД. CODASYL, яка пізніше взята за основу IMS — власної розробки IBM.

1980-ті рр. поява перших комерційних версій реляційних БД Oracle та DB2. Реляційні БД починають успішно витісняти мережеві та ієрархічні. Дослідження децентралізованих (розподілених) систем БД, проте вони не відіграють особливої ролі на ринку БД.

1990-ті рр. увага науковців спрямовується на об'єктно-орієнтовані БД, які знайшли застосування в першу чергу в тих галузях, де використовуються комплексні дані: інженерні, мультимедійні БД.

1.2 Основні види баз даних

Існує кілька різновидів програмної реалізації баз даних, які використовуються для зберігання та обробки інформації:

Файлові системи: Найпростішим видом картотеки є зберігання даних у вигляді файлів на диску. Цей підхід має низьку вартість, але обмежену функціональність і швидкість доступу до даних.

Реляційні бази даних: Більш складний та ефективний спосіб зберігання даних – використання реляційних баз даних (RDBMS), таких як MySQL, PostgreSQL, Mongo DB що дозволяють зберігати великі обсяги інформації і швидко виконувати запити.

Об'єктно-орієнтовані бази даних: Цей підхід дозволяє зберігати дані у вигляді об'єктів, що є особливо зручним при використанні об'єктно- орієнтованого програмування (ООП).

Хмарні рішення: З розвитком хмарних технологій стало можливим використовувати хмарні бази даних для зберігання та обробки інформації. Це дозволяє забезпечити високу доступність та масштабованість системи.

1.3 Висновки за розділом

Проаналізувавши технології та різновиди баз даних, було прийнято рішення розробити веб-додаток для керування медіаколекцією з використанням реляційної бази даних Mongo DB.

РОЗДІЛ 2

ВИБІР ТА ОБГРУНТУВАННЯ ПРОГРАМНО-АПАРАТНИХ РІШЕНЬ

2.1 Мова програмування TypeScript

TypeScript — мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки вебзастосунків, що розширює можливості JavaScript

Розробником мови TypeScript є Андерс Гейлсберг, який створив раніше C#, Turbo Pascal і Delphi.

Переваги над JavaScript:

- можливість явного визначення типів (статична типізація),
- підтримка використання повноцінних класів (як у традиційних об'єктно-орієнтованих мовах),
- підтримка підключення модулів.

За задумом ці нововведення мають підвищити швидкість розробки, прочитність, рефакторинг і повторне використання коду, здійснювати пошук помилок на етапі розробки та компіляції, а також швидкодію програм.

2.2 Фреймворк Angular

Angular — це безкоштовний фреймворк односторінкових веб-додатків на основі TypeScript з відкритим вихідним кодом, що працює на Node.js. Його розробкою займається Angular Team у складі Google, а також спільнота приватних осіб і корпорацій. Angular повністю переписаний тією ж командою, яка створила AngularJS. Екосистема Angular складається з різноманітної групи з понад 1.7 мільйона розробників, авторів бібліотек та творців контенту.

2.3 Візуальна середа розробки

Visual Studio Code — який також зазвичай називають VS Code — це редактор початкового коду, створений Microsoft із Electron Framework для Windows, Linux і macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність.

В опитуванні розробників Stack Overflow 2022 серед 71 010 респондентів Visual Studio Code назвали найпопулярнішим інструментом середовища розробника, при цьому 74,48 % повідомили, що вони ним користуються.

Рисунок 2.1— Інтерфейс програми VS Code

2.4 Висновки за розділом

Розглянувши засоби програмування, було прийнято рішення створювати проект за допомогою Angular у середовищі VS Code.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ

Щоб створити веб-додаток для керування медіаколекцією, потрібно розробити систему завантаження та зчитування даних, яка буде працювати врежимі реального часу. Також необхідна база даних, де будуть зберігатися усі дані, такі, як інформація про треки та самі треки.

3.1 Архітектура програми

Для початку потрібно розділити розробку програми на 2 частини: клієнтську і серверну. Потрібно буде працювати з цими частинами, щоб зробитипрацюючий веб-додаток.

Щоб краще розуміти, що повинно бути імплементовано в програму і як це повинно взаємодіяти, були створені UML діаграми: прецедентів, взаємодій, діяльності, станів, взаємовідносин ролей та класів. Види UMLдіаграм відображені на рисунках 3.1 – 3.8.

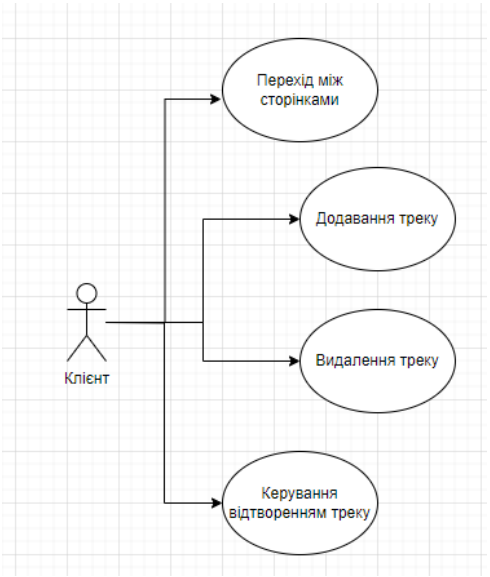


Рисунок 3.1— Діаграма прецедентів

Існує лише один актор додатку – користувач. Він може переглядати список,а також завантажувати, керувати та видаляти треки.

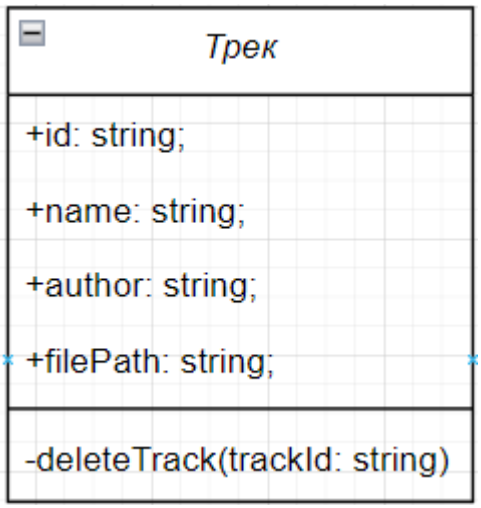


Рисунок 3.2— Діаграма класів

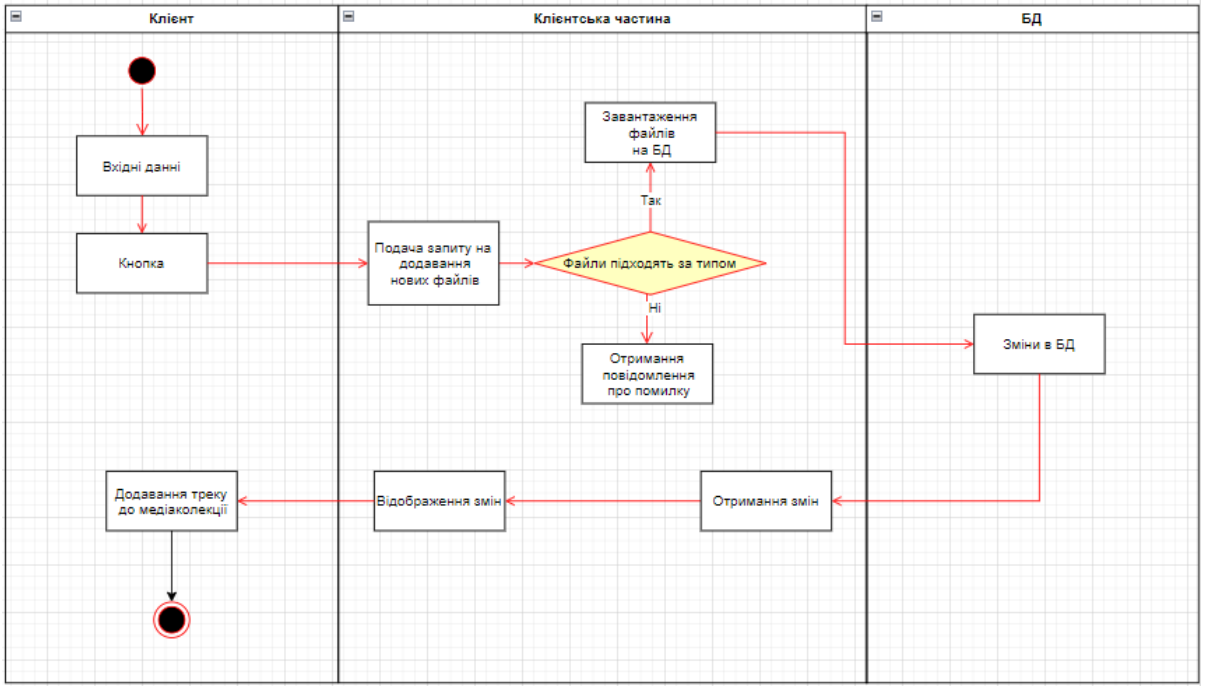


Рисунок 3.3— Діаграма діяльності для додавання треку

3.2 Технологічний комплекс

Під час створення цього додатка були використані наступні інструменти:

- Бібліотека Angular ;
- Мови програмування: TypeScript, JavaScript, CSS, HTML.

На рисунку 3.4 приклад веб-додатку на Angular.

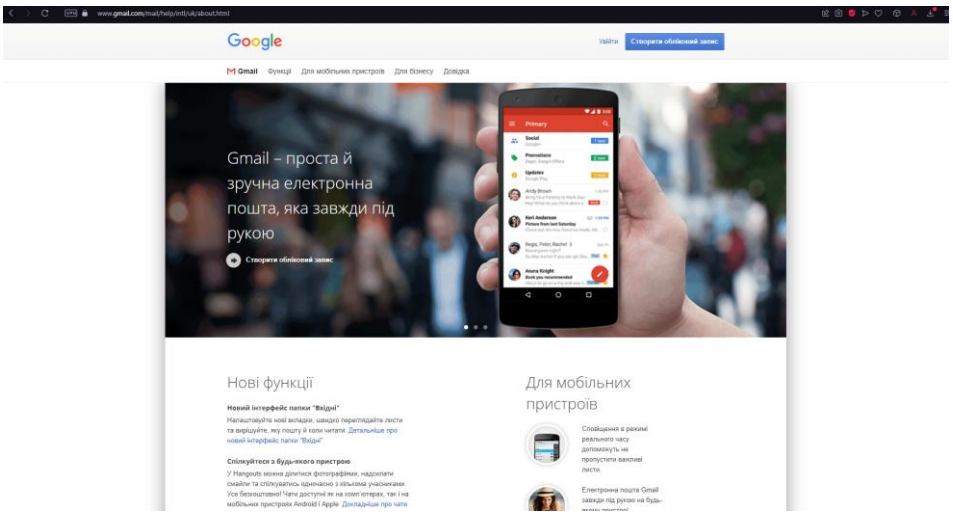


Рисунок 3.4— приклад веб-додатку на Angular

3.3 Організація середовища для розробки ПЗ

По-перше, потрібно організувати середу розробки ПЗ. Щоб встановити Angular треба встановити Node.js та npm:

В консолі:

npm install

Завантажити та встановити останню LTS-версію Node.js з офіційного сайту.

Перевірити встановлення за допомогою команди:

node -v

3.4 Структура веб-додатку

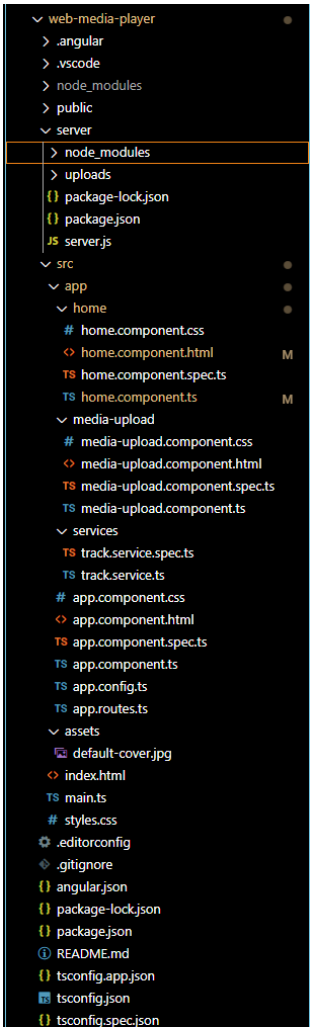
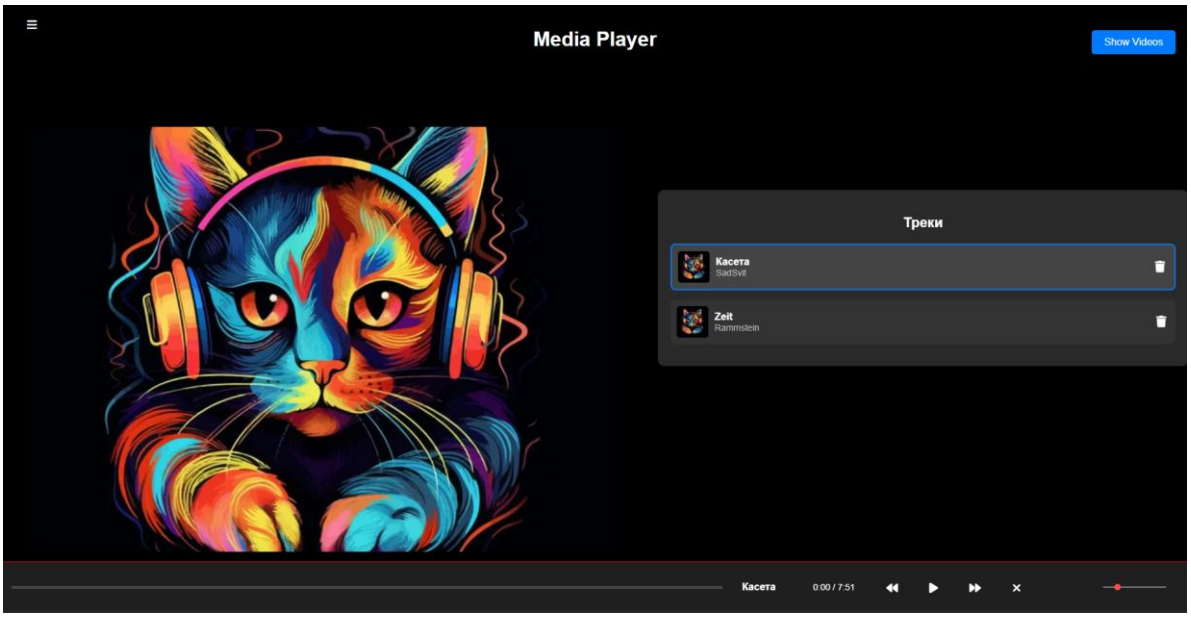


Рис.3.5— Структура програми

3.5 Клієнтська частина

Клієнтська частина реалізована в файлах home.component(Додадок А), media-upload.component(Додаток Б) та track.service(Додаток В).

home.component для відображення головної сторінки



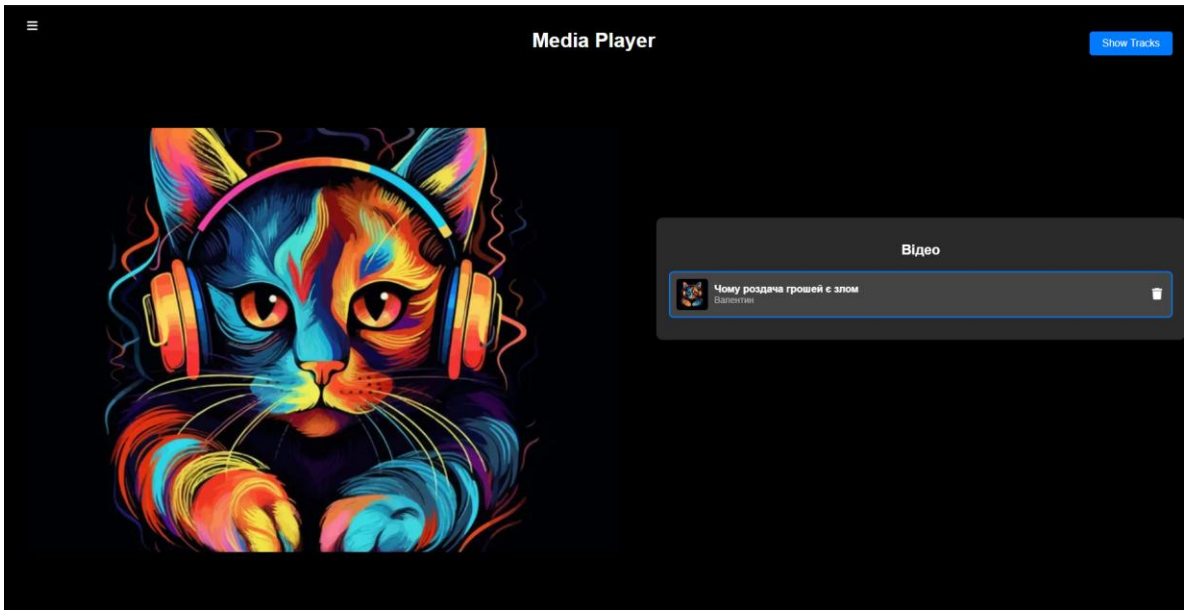


Рис.3.6— Головна сторінка

media-upload.component для сторінки завантаження файлів

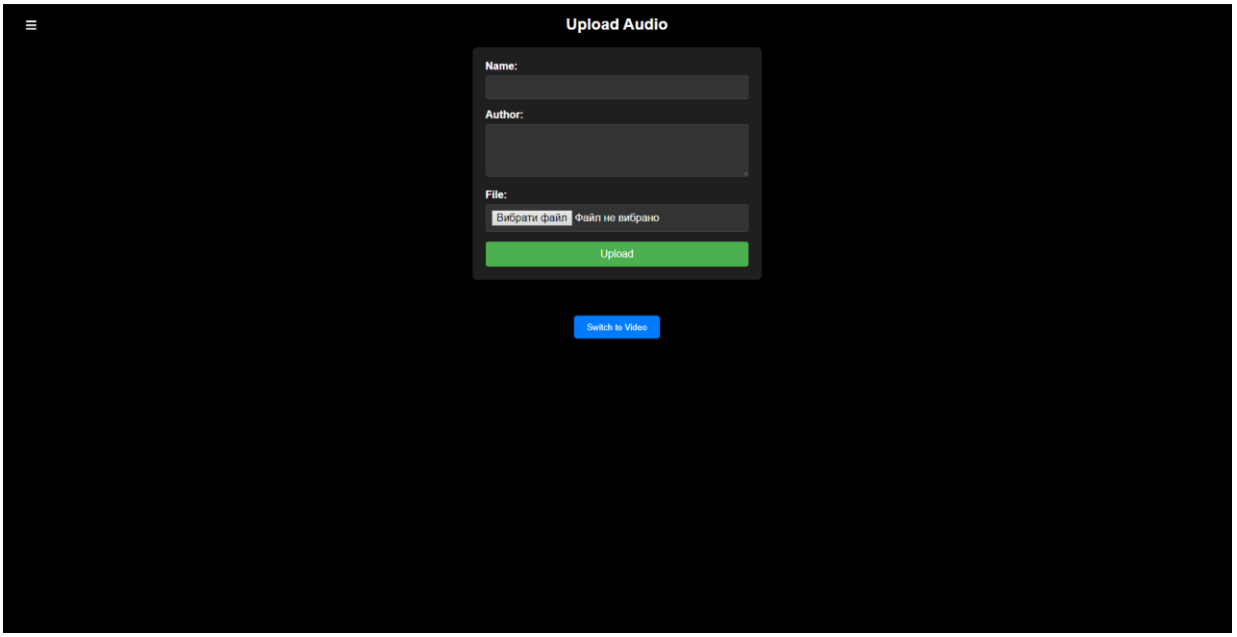


Рис.3.7— Сторінка для завантаження файлів

3.6 Висновки за розділом

Було створено веб-додаток для керування медіаколекцією.

ВИСНОВКИ

У ході виконання курсової роботи було розроблено веб-додаток для керування медіаколекцією з використанням фреймворку Angular та бази даних Mongo DB.

Основними завданнями, які були вирішені, є:

- Аналіз предметної області та огляд існуючих аналогів. Було проведено дослідження існуючих рішень для керування медіафайлами, що дозволило визначити ключові функціональні вимоги до розроблюваного додатку.
- Вибір та обґрунтування програмних засобів та технологій. Було обрано фреймворк Angular для розробки клієнтської частини додатку, а також Mongo DB як реляційну базу даних для зберігання медіафайлів. Також було використано Visual Studio Code як середовище розробки.
- Реалізація проекту. Було розроблено архітектуру додатку, яка включає клієнтську та серверну частини. Клієнтська частина реалізована на Angular, а серверна — на Node.js. Було створено функціонал для завантаження, пошуку та видалення медіафайлів.
- Тестування та налаштування. Було проведено тестування додатку, що дозволило виявити та виправити помилки, а також оптимізувати продуктивність.

Результатом роботи став функціональний веб-додаток, який дозволяє користувачам ефективно керувати своєю медіаколекцією. Додаток забезпечує зручний інтерфейс для завантаження, перегляду та видалення медіафайлів, а також підтримує різні типи файлів.

У майбутньому можуть бути додані наступні функції для покращення додатку:

- Редагування метаданих файлів.
- Підтримка хмарних рішень для зберігання даних.
- Інтеграція з соціальними мережами для обміну медіафайлами.

Таким чином, розроблений додаток є важливим кроком у розвитку систем керування медіаданими, що дозволяє користувачам ефективно організовувати та керувати своєю колекцією файлів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний портал фреймворку Angular [Електронний ресурс] / Режим доступу URL: https://angular.dev
2. Офіційний портал Stackoverflow[Електронний ресурс] / Режим доступу URL: https://stackoverflow.com/.
3. Офіційний портал для створення UML [Електронний ресурс] / Режим доступу URL: https://app.diagrams.net

ДОДАТОК

КОД ПРОГРАММИ

Додаток А(home.component.ts, home.component.html)

home.component.ts

```
import { Component, OnInit, AfterViewInit, ViewChild, ElementRef } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
import { MediaService, Track, Video } from '../services/track.service';

@Component({
  selector: 'app-home',
  standalone: true,
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  imports: [CommonModule, RouterModule, HttpClientModule],
})
export class HomeComponent implements OnInit, AfterViewInit {
  isMenuOpen: boolean = false;
  tracks: Track[] = [];
  videos: Video[] = [];
  filteredTracks: Track[] = [];
  currentTrack: Track | null = null;
  currentVideo: Video | null = null;
  audioElement?: HTMLAudioElement;
  isPlaying: boolean = false;
  repeatMode: 'none' | 'single' | 'all' = 'none';
  currentTime: string = '0:00';
  duration: string = '0:00';
  currentTimeInSeconds: number = 0;
  durationInSeconds: number = 0;
  showTracks: boolean = true;
  showControls: boolean = true;
  showVideoPlayer: boolean = false;

  @ViewChild('videoPlayer', { static: false }) videoPlayer!: ElementRef<HTMLVideoElement>;

  constructor(private trackService: MediaService) {}

  ngOnInit(): void {
    this.loadTracks();
    this.loadVideos();
  }

  ngAfterViewInit(): void {
    this.audioElement = new Audio();
    this.audioElement.style.display = 'none';
    document.body.appendChild(this.audioElement);

    this.audioElement.addEventListener('ended', () => {
      if (this.repeatMode === 'single') {
        this.audioElement!.currentTime = 0;
        this.audioElement!.play();
      } else {
        this.playNextTrack();
      }
    });

    this.audioElement.ontimeupdate = () => {
      this.updateTime(this.audioElement!);
    };

    this.audioElement.onloadedmetadata = () => {
      this.durationInSeconds = this.audioElement!.duration;
      this.duration = this.formatTime(this.audioElement!.duration);
    };

    this.videoPlayer.nativeElement.addEventListener('ended', () => {
      if (this.repeatMode === 'single') {
        this.videoPlayer.nativeElement.currentTime = 0;
        this.videoPlayer.nativeElement.play();
      } else {
        this.playNextVideo();
      }
    });

    this.videoPlayer.nativeElement.ontimeupdate = () => {
      this.updateTime(this.videoPlayer.nativeElement);
```

```
  };

  this.videoPlayer.nativeElement.onloadedmetadata = () => {
    this.durationInSeconds = this.videoPlayer.nativeElement.duration;
    this.duration = this.formatTime(this.videoPlayer.nativeElement.duration);
  };
}

toggleMenu(): void {
  this.isMenuOpen = !this.isMenuOpen;
}

loadTracks(): void {
  this.trackService.getTracks().subscribe(
    (data: any) => {
      this.tracks = data.map((track: any) => ({
        id: track.id,
        name: track.name,
        author: track.author,
        filePath: track.filePath,
      }));

      console.log('Loaded tracks:', this.tracks);

      if (this.tracks.length > 0) {
        this.setCurrentTrack(this.tracks[0], false);
      }

      this.filteredTracks = [...this.tracks];
    },
    (error) => {
      console.error('Error loading tracks:', error);
    }
  );
}

loadVideos(): void {
  this.trackService.getVideos().subscribe(
    (data: any) => {
      this.videos = data.map((video: any) => ({
        id: video.id,
        name: video.name,
        author: video.author,
        filePath: video.filePath,
      }));

      console.log('Loaded videos:', this.videos);

      if (this.videos.length > 0) {
        this.setCurrentVideo(this.videos[0], false);
      }
    },
    (error) => {
      console.error('Error loading videos:', error);
    }
  );
}

togglePlay(media: Track | Video): void {
  if (media.hasOwnProperty('filePath')) {
    if (this.isTrack(media)) {
      if (this.currentTrack && this.currentTrack === media) {
        this.togglePlayPause();
      } else {
        this.setCurrentTrack(media as Track, true);
      }
    } else if (this.isVideo(media)) {
      if (this.currentVideo && this.currentVideo === media) {
        this.togglePlayPauseVideo();
      } else {
        this.setCurrentVideo(media as Video, true);
      }
    }
  }
}


private isTrack(media: Track | Video): media is Track {
  return (media as Track).filePath !== undefined;
}
```

```
private isVideo(media: Track | Video): media is Video {
  return (media as Video).filePath !== undefined;
}

togglePlayPause(): void {
  if (this.audioElement) {
    if (this.audioElement.paused) {
      this.audioElement.play();
      this.isPlaying = true;
      this.showControls = true;
      this.showVideoPlayer = false;
      if (this.videoPlayer) {
        this.videoPlayer.nativeElement.pause();
      }
    } else {
      this.audioElement.pause();
      this.isPlaying = false;
    }
  }
}

togglePlayPauseVideo(): void {
  if (this.videoPlayer) {
    if (this.videoPlayer.nativeElement.paused) {
      this.videoPlayer.nativeElement.play();
      this.isPlaying = true;
      this.showControls = false;
      this.showVideoPlayer = true;
    } else {
      this.videoPlayer.nativeElement.pause();
      this.isPlaying = false;
    }
  }
}

playNextTrack(): void {
  if (this.currentTrack && this.tracks.length > 0) {
    const currentIndex = this.tracks.findIndex((track) => track === this.currentTrack);
    const nextIndex = (currentIndex + 1) % this.tracks.length;
    this.setCurrentTrack(this.tracks[nextIndex]);
  } else if (this.currentVideo && this.videos.length > 0) {
    const currentIndex = this.videos.findIndex((video) => video === this.currentVideo);
    const nextIndex = (currentIndex + 1) % this.videos.length;
    this.setCurrentVideo(this.videos[nextIndex]);
  }
}

playPreviousTrack(): void {
  if (this.currentTrack && this.tracks.length > 0) {
    const currentIndex = this.tracks.findIndex((track) => track === this.currentTrack);
    const prevIndex = (currentIndex - 1 + this.tracks.length) % this.tracks.length;
    this.setCurrentTrack(this.tracks[prevIndex]);
  } else if (this.currentVideo && this.videos.length > 0) {
    const currentIndex = this.videos.findIndex((video) => video === this.currentVideo);
    const prevIndex = (currentIndex - 1 + this.videos.length) % this.videos.length;
    this.setCurrentVideo(this.videos[prevIndex]);
  }
}

playNextVideo(): void {
  if (this.currentVideo && this.videos.length > 0) {
    const currentIndex = this.videos.findIndex((video) => video === this.currentVideo);
    const nextIndex = (currentIndex + 1) % this.videos.length;
    this.setCurrentVideo(this.videos[nextIndex]);
  }
}

toggleRepeatMode() {
  if (this.repeatMode === 'none') {
    this.repeatMode = 'single';
  } else if (this.repeatMode === 'single') {
    this.repeatMode = 'all';
  } else {
    this.repeatMode = 'none';
  }
}

setVolume(event: Event): void {
  const inputElement = event.target as HTMLInputElement;
  if (inputElement) {
    const value = parseFloat(inputElement.value);
```

```
    if (this.audioElement) {
      this.audioElement.volume = value;
    }
    if (this.videoPlayer) {
      this.videoPlayer.nativeElement.volume = value;
    }
  }
}

seekTrack(event: MouseEvent): void {
  const sliderContainer = event.currentTarget as HTMLElement;
  const rect = sliderContainer.getBoundingClientRect();
  const offsetX = event.clientX - rect.left;
  const percentage = Math.min(1, Math.max(0, offsetX / rect.width));

  if (this.audioElement) {
    this.audioElement.currentTime = percentage * this.durationInSeconds;
  }
  if (this.videoPlayer) {
    this.videoPlayer.nativeElement.currentTime = percentage * this.durationInSeconds;
  }
}

private setCurrentTrack(track: Track, autoPlay: boolean = true): void {
  if (!this.audioElement) {
    console.error('Audio element not found!');
    return;
  }

  this.currentTrack = track;
  const fixedFilePath = track.filePath.replace(/\\/g, '/');
  this.audioElement.src = `http://localhost:5000/${fixedFilePath}`;
  this.audioElement.load();

  if (autoPlay) {
    this.audioElement.play();
    this.isPlaying = true;
    this.showControls = true;
    this.showVideoPlayer = false;
    if (this.videoPlayer) {
      this.videoPlayer.nativeElement.pause();
    }
  } else {
    this.isPlaying = false;
  }
  this.currentTime = '0:00';
  this.audioElement.addEventListener('loadedmetadata', () => {
    this.duration = this.formatTime(this.audioElement!.duration);
  });
}

private setCurrentVideo(video: Video, autoPlay: boolean = true): void {
  if (!this.videoPlayer) {
    console.error('Video player not found!');
    return;
  }

  this.currentVideo = video;
  const fixedFilePath = video.filePath.replace(/\\/g, '/');
  this.videoPlayer.nativeElement.src = `http://localhost:5000/${fixedFilePath}`;
  this.videoPlayer.nativeElement.load();

  if (autoPlay) {
    this.videoPlayer.nativeElement.play();
    this.isPlaying = true;
    this.showControls = false;
    this.showVideoPlayer = true;
  } else {
    this.isPlaying = false;
  }
  this.currentTime = '0:00';
  this.videoPlayer.nativeElement.addEventListener('loadedmetadata', () => {
    this.duration = this.formatTime(this.videoPlayer.nativeElement.duration);
  });
}

deleteTrack(track: Track): void {
  if (!track.id) {
    console.error('Track ID is undefined');
    return;
  }
```

```
  this.trackService.deleteTrack(track.id).subscribe(
    () => {
      this.tracks = this.tracks.filter((t) => t.id !== track.id);
      if (this.currentTrack && this.currentTrack.id === track.id) {
        if (this.audioElement) {
          this.audioElement.pause();
          this.audioElement.src = '';
        }
        this.currentTrack = null;
        this.isPlaying = false;
      }

      this.loadTracks();
    },
    (error) => {
      console.error('Error deleting track:', error);
    }
  );
 }

 deleteVideo(video: Video): void {
  if (!video.id) {
    console.error('Video ID is undefined');
    return;
  }

  this.trackService.deleteVideo(video.id).subscribe(
    () => {
      this.videos = this.videos.filter((v) => v.id !== video.id);
      if (this.currentVideo && this.currentVideo.id === video.id) {
        if (this.videoPlayer) {
          this.videoPlayer.nativeElement.pause();
          this.videoPlayer.nativeElement.src = '';
        }
        this.currentVideo = null;
        this.isPlaying = false;
      }

      this.loadVideos();
    },
    (error) => {
      console.error('Error deleting video:', error);
    }
  );
 }

 updateTime(media: HTMLAudioElement | HTMLVideoElement): void {
  this.currentTime = this.formatTime(media.currentTime);
  this.currentTimeInSeconds = media.currentTime;
 }

 formatTime(time: number): string {
  const minutes = Math.floor(time / 60);
  const seconds = Math.floor(time % 60);
  return `${minutes}:${seconds < 10 ? '0' + seconds : seconds}`;
 }

 toggleMediaList(): void {
  this.showTracks = !this.showTracks;

  if (!this.showTracks) {
    this.showControls = false;
  } else {
    this.showControls = true;
  }
 }

 openVideoInNewTab(video: Video): void {
  const videoUrl = `http://localhost:5000/${video.filePath.replace(/\\/g, '/')}`;
  window.open(videoUrl, '_blank');
 }
}
```

home.component.html

```
<h1>Media Player</h1>

<div class="main-container">
<div class="media-container">
<video #videoPlayer class="media-video" [class.hidden]="!showVideoPlayer" controls></video>
<img *ngIf="!showVideoPlayer" src="https://st5.depositphotos.com/15985744/65632/v/450/depositphotos_656321876-stock-illustration-colorful-vector-poster-cat-listening.jpg" alt="Placeholder Image" class="media-image" />
</div>

<div class="controls-container" [class.hidden]="!showControls">
<div class="slider-container" (click)="seekTrack($event)">
<div class="slider-bar" [style.width.%]="(currentTimeInSeconds / durationInSeconds) * 100"></div>
<div class="slider-knob-container">
<div class="slider-knob" [style.left.%]="(currentTimeInSeconds / durationInSeconds) * 100"></div>
</div>
</div>

<div class="track-info">
<span class="track-title">{{ currentTrack?.name || currentVideo?.name }}</span>
</div>
<div class="track-time">
<span>{{ currentTime }}</span> / <span>{{ duration }}</span>
</div>

<button (click)="playPreviousTrack()">
<i class="fa fa-backward"></i>
</button>
<button (click)="togglePlayPause()">
<i [class]="isPlaying ? 'fa fa-pause' : 'fa fa-play'"></i>
</button>
<button (click)="playNextTrack()">
<i class="fa fa-forward"></i>
</button>
<button (click)="toggleRepeatMode()">
<i [ngClass]="{
'fa fa-times': repeatMode === 'none',
'fa-solid fa-1': repeatMode === 'single',
'fa-solid fa-repeat': repeatMode === 'all'
}"></i>
</button>

<input
id="volumeSlider"
type="range"
min="0"
max="1"
step="0.01"
value="0.2"
(input)="setVolume($event)">
</div>

<div class="main-container" [class.menu-open]="isMenuOpen">
<div id="menuPanel" class="menu-panel" [class.open]="isMenuOpen">
<div class="menu-buttons">
<button class="menu-item" routerLink="/">Home</button>
<button class="menu-item" routerLink="/upload">Upload</button>
</div>
</div>

<button id="menuButton" class="menu-button" (click)="toggleMenu()">
<i class="fa fa-bars"></i>
</button>
</div>

<router-outlet></router-outlet>

<button class="toggle-media-list" (click)="toggleMediaList()">
{{ showTracks ? 'Show Videos' : 'Show Tracks' }}
</button>

<!-- Список треків -->
<div class="track-list" *ngIf="showTracks">
<h2>Треки</h2>
<div *ngFor="let track of tracks"
class="track-item"
[ngClass]="{'active': track === currentTrack}">
<div class="track-cover-container">
<img src="https://st5.depositphotos.com/15985744/65632/v/450/depositphotos_656321876-stock-illustration-colorful-vector-poster-cat-listening.jpg"
class="track-cover" />
<button class="play-button" (click)="togglePlay(track)">
```

```
<i class="fa fa-play"></i>
</button>
</div>
<div class="track-info">
<p class="track-title">{{ track.name }}</p>
<p class="track-artist">{{ track.author }}</p>
</div>
<button class="delete-button" (click)="deleteTrack(track)">
<i class="fa fa-trash"></i>
</button>
</div>
</div>

<!-- Список відео -->
<div class="video-list" *ngIf="!showTracks">
<h2>Відео</h2>
<div *ngFor="let video of videos" class="video-item" [ngClass]="{'active': video === currentVideo}">
<div class="video-cover-container">
<img src="https://st5.depositphotos.com/15985744/65632/v/450/depositphotos_656321876-stock-illustration-colorful-vector-poster-cat-listening.jpg" class="track-cover" />
<button class="play-button" (click)="openVideoInNewTab(video)">
<i class="fa fa-play"></i>
</button>
</div>
<div class="video-info">
<p class="video-title">{{ video.name }}</p>
<p class="video-artist">{{ video.author }}</p>
</div>
<button class="delete-button" (click)="deleteVideo(video)">
<i class="fa fa-trash"></i>
</button>
</div>
</div>
</div>
```

Додаток Б(media-upload.component.ts, media-upload.component.html)

media-upload.component.ts

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClient } from '@angular/common/http';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-media-upload',
  standalone: true,
  templateUrl: './media-upload.component.html',
  styleUrls: ['./media-upload.component.css'],
  imports: [FormsModule, RouterModule],
})
export class MediaUploadComponent {
  uploadType: 'audio' | 'video' = 'audio';
  name: string = '';
  author: string = '';
  file: File | null = null;
  isMenuOpen: boolean = false;

  constructor(private http: HttpClient) {}

  toggleUploadType() {
    this.uploadType = this.uploadType === 'audio' ? 'video' : 'audio';
  }

  onFileChange(event: any) {
    const file = event.target.files[0];
    const allowedTypes =
      this.uploadType === 'audio'
        ? ['audio/mpeg', 'audio/wav']
        : ['video/mp4', 'video/webm'];

    if (file && allowedTypes.includes(file.type)) {
      this.file = file;
    } else {
      alert(
        `Please select a valid ${
          this.uploadType === 'audio' ? 'audio' : 'video'
        } file.`
      );
      this.file = null;
    }
  }

  toggleMenu(): void {
    this.isMenuOpen = !this.isMenuOpen;
  }

  uploadFile() {
    if (!this.file || !this.name || !this.author) {
      alert('Please fill in all fields and select a file.');
      return;
    }

    const formData = new FormData();
    formData.append('file', this.file);
    formData.append('name', this.name);
    formData.append('author', this.author);

    const apiUrl =
      this.uploadType === 'audio'
        ? 'http://localhost:5000/uploads'
        : 'http://localhost:5000/uploads/video';

    this.http.post(apiUrl, formData).subscribe({
      next: (response) => {
        console.log('Upload successful', response);
        alert('File uploaded successfully!');
        this.resetForm();
      },
      error: (err) => {
        console.error('Upload error', err);
        alert('File upload failed.');
      },
    });
  }
```

```
  resetForm() {
    this.name = '';
    this.author = '';
    this.file = null;
  }
}
```

media-upload.component.html

```
<div>
  <button class="toggle-button" (click)="toggleUploadType()">
    {{ uploadType === 'audio' ? 'Switch to Video' : 'Switch to Audio' }}
  </button>

  <h1>Upload {{ uploadType === 'audio' ? 'Audio' : 'Video' }}</h1>
  <form (submit)="uploadFile()">
    <div>
      <label>Name:</label>
      <input type="text" [(ngModel)]="name" name="name" required />
    </div>
    <div>
      <label>Author:</label>
      <textarea [(ngModel)]="author" name="author"></textarea>
    </div>
    <div>
      <label>File:</label>
      <input type="file" (change)="onFileChange($event)" />
    </div>
    <button type="submit">Upload</button>
  </form>
</div>

<div class="main-container" [class.menu-open]="isMenuOpen">
  <div id="menuPanel" class="menu-panel" [class.open]="isMenuOpen">
    <div class="menu-buttons">
      <button class="menu-item" routerLink="/">Home</button>
      <button class="menu-item" routerLink="/upload">Upload</button>
    </div>
  </div>

  <button id="menuButton" class="menu-button" (click)="toggleMenu()">
    <i class="fa fa-bars"></i>
  </button>
</div>

<router-outlet></router-outlet>
```

Додаток В(track.service.ts)

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';

// Інтерфейс для треку
export interface Track {
  id: string;
  name: string;
  author: string;
  filePath: string;
}

// Інтерфейс для відео
export interface Video {
  id: string;
  name: string;
  author: string;
  filePath: string;
}

@Injectable({
  providedIn: 'root',
})
export class MediaService {
  private apiUrl = 'http://localhost:5000/uploads'; // Базовий URL для треків
  private videoApiUrl = 'http://localhost:5000/uploads/video'; // Базовий URL для відео

  constructor(private http: HttpClient) {}

  // Отримання списку треків
  getTracks(): Observable<Track[]> {
    return this.http.get<Track[]>(this.apiUrl).pipe(
      tap(tracks => {
        console.log('Fetched tracks:', tracks);
      }),
      catchError(this.handleError)
    );
  }

  // Видалення треку
  deleteTrack(trackId: string): Observable<any> {
    if (!trackId) {
      console.error('Track ID is undefined or null');
      return throwError(() => new Error('Track ID is undefined or null'));
    }

    return this.http.delete(`${this.apiUrl}/${trackId}`).pipe(
      tap(() => {
        console.log(`Track with ID ${trackId} deleted successfully`);
      }),
      catchError(this.handleError)
    );
  }

  // Отримання списку відео
  getVideos(): Observable<Video[]> {
    return this.http.get<Video[]>(this.videoApiUrl).pipe(
      tap(videos => {
        console.log('Fetched videos:', videos);
      }),
      catchError(this.handleError)
    );
  }

  // Видалення відео
  deleteVideo(videoId: string): Observable<any> {
    if (!videoId) {
      console.error('Video ID is undefined or null');
      return throwError(() => new Error('Video ID is undefined or null'));
    }

    return this.http.delete(`${this.videoApiUrl}/${videoId}`).pipe(
      tap(() => {
        console.log(`Video with ID ${videoId} deleted successfully`);
      }),
      catchError(this.handleError)
    );
  }
```

```
  // Загальний обробник помилок
  private handleError(error: HttpErrorResponse) {
    console.error('An error occurred:', error);
    return throwError(() => new Error(error.message || 'Server error'));
  }
}
```