Un'attività spesso richiesta, nei progetti su cui lavoro, è quella di gestire l’upload e il download di file in Angular. Ci sono diversi modi per poter sviluppare queste funzionalità: l'approccio migliore dipende spesso dalle API che si hanno a dispozione.

Supponiamo di avere delle API scritte in Asp.Net Core, nello specifico un controller con tre action :

* **Upload**, per ricevere un file e salvarlo nella cartella ./wwwroot/upload;
* **Download**, per recuperare un file dalla cartella ./wwwroot/upload;
* **Files**, per ottenere la lista dei file presenti in ./wwwroot/upload;

Una possibile implementazione del controller è la seguente:

namespace BackEnd.Controllers

{

   [Route("api")]

   [ApiController]

   public class UploadDownloadController: ControllerBase

   {

       private IHostingEnvironment \_hostingEnvironment;

       public UploadDownloadController(IHostingEnvironment environment) {

           \_hostingEnvironment = environment;

       }

       [HttpPost]

       [Route("upload")]

       public async Task<iactionresult> Upload(IFormFile file)

       {

           var uploads = Path.Combine(\_hostingEnvironment.WebRootPath, "uploads");

           if(!Directory.Exists(uploads))

           {

               Directory.CreateDirectory(uploads);

           }

           if (file.Length > 0) {

               var filePath = Path.Combine(uploads, file.FileName);

               using (var fileStream = new FileStream(filePath, FileMode.Create)) {

                   await file.CopyToAsync(fileStream);

               }

           }

           return Ok();

       }

       [HttpGet]

       [Route("download")]

       public async Task<iactionresult> Download([FromQuery] string file) {

           var uploads = Path.Combine(\_hostingEnvironment.WebRootPath, "uploads");

           var filePath = Path.Combine(uploads, file);

           if (!System.IO.File.Exists(filePath))

               return NotFound();

           var memory = new MemoryStream();

           using (var stream = new FileStream(filePath, FileMode.Open))

           {

               await stream.CopyToAsync(memory);

           }

           memory.Position = 0;

           return File(memory, GetContentType(filePath), file);

       }

       [HttpGet]

       [Route("files")]

       public IActionResult Files() {

           var result =  new List<string>();

           var uploads = Path.Combine(\_hostingEnvironment.WebRootPath, "uploads");

           if(Directory.Exists(uploads))

           {

               var provider = \_hostingEnvironment.ContentRootFileProvider;

               foreach (string fileName in Directory.GetFiles(uploads))

               {

                   var fileInfo = provider.GetFileInfo(fileName);

                   result.Add(fileInfo.Name);

               }

           }

           return Ok(result);

       }

       private string GetContentType(string path)

       {

           var provider = new FileExtensionContentTypeProvider();

           string contentType;

           if(!provider.TryGetContentType(path, out contentType))

           {

               contentType = "application/octet-stream";

           }

           return contentType;

       }

   }

}

Come potete vedere, niente di complesso.  Interessante il *FileExtensionContentTypeProvider* di .Net Core che vi permette di ricavare il content type dall'estensione del file.

A questo punto possiamo creare un progetto Angular con la CLI, con il quale vogliamo caricare dei file, visualizzarli e farne il download. In più, mostreremo lo stato di avanzamento durante il download o l’upload, sfruttando una delle funzionalità del nuovo HttpClient di Angular. Creeremo un componente specifico per ogni operazione, Upload e Download, che utilizzeremo da un componente FileManager che mostrerà la lista dei file caricati.

Tutti e tre i componenti condivideranno un service in cui implementeremo le chiamate HTTP alla nostre API:

import { Injectable } from '@angular/core';

import { HttpClient, HttpRequest, HttpEvent, HttpResponse } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable()

export class UploadDownloadService {

  private baseApiUrl: string;

  private apiDownloadUrl: string;

  private apiUploadUrl: string;

  private apiFileUrl: string;

  constructor(private httpClient: HttpClient) {

    this.baseApiUrl = '<http://localhost:5001/api/>';

    this.apiDownloadUrl = this.baseApiUrl + 'download';

    this.apiUploadUrl = this.baseApiUrl + 'upload';

    this.apiFileUrl = this.baseApiUrl + 'files';

  }

  public downloadFile(file: string): Observable<HttpEvent<Blob>> {

    return this.httpClient.request(new HttpRequest(

      'GET',

      `${this.apiDownloadUrl}?file=${file}`,

      null,

      {

        reportProgress: true,

        responseType: 'blob'

      }));

  }

  public uploadFile(file: Blob): Observable<HttpEvent<void>> {

    const formData = new FormData();

    formData.append('file', file);

    return this.httpClient.request(new HttpRequest(

      'POST',

      this.apiUploadUrl,

      formData,

      {

        reportProgress: true

      }));

  }

  public getFiles(): Observable<string[]> {

    return this.httpClient.get<string[]>(this.apiFileUrl);

  }

Rispetto al classico uso che si fa di HttpClient, che potete vedere nel metodo getFiles(), in dowloadFile() e uploadFile(), utilizziamo il metodo request(), che ci permette di specificare una HttpRequest con tutte le sue opzioni, tra cui l'opzione reportProgress impostata su true. Questa opzione ci abilita a ricevere gli aggiornamenti sullo stato dello scambio dati tra client e server. Come? Guardiamolo nel nostro componente Upload:

import { Component, Output, EventEmitter, Input, ViewChild, ElementRef } from '@angular/core';

import { UploadDownloadService } from 'src/app/services/upload-download.service';

import { HttpEventType } from '@angular/common/http';

import { ProgressStatus, ProgressStatusEnum } from 'src/app/models/progress-status.model';

@Component({

  selector: 'app-upload',

  templateUrl: 'upload.component.html'

})

export class UploadComponent {

  @Input() public disabled: boolean;

  @Output() public uploadStatus: EventEmitter<progressstatus>;

  @ViewChild('inputFile') inputFile: ElementRef;

  constructor(private service: UploadDownloadService) {

    this.uploadStatus = new EventEmitter<ProgressStatus>();

  }

  public upload(event) {

    if (event.target.files && event.target.files.length > 0) {

      const file = event.target.files[0];

      this.uploadStatus.emit({status: ProgressStatusEnum.START});

      this.service.uploadFile(file).subscribe(

        data => {

          if (data) {

            switch (data.type) {

              case HttpEventType.UploadProgress:

                this.uploadStatus.emit( {status: ProgressStatusEnum.IN\_PROGRESS, percentage: Math.round((data.loaded / data.total) \* 100)});

                break;

              case HttpEventType.Response:

                this.inputFile.nativeElement.value = '';

                this.uploadStatus.emit( {status: ProgressStatusEnum.COMPLETE});

                break;

            }

          }

        },

        error => {

          this.inputFile.nativeElement.value = '';

          this.uploadStatus.emit({status: ProgressStatusEnum.ERROR});

        }

      );

    }

  }

}

Come potete vedere, la sottoscrizione all'observable restituito dal HttpClient ci fornisce un oggetto di tipo HttpEvent, la cui proprietà *type* può assumere uno di cinque valori:

* **Sent**: quando la richiesta è stata inviata;
* **UploadProgress**: quando è in corso l'upload;
* **ResponseHeader**: quando status code e intestazioni sono state ricevute;
* **DownloadProgress**: quando è in corso il download;
* **Response**: quando l'intera risposta è stata ricevuta;
* **User**: quando viene sollevato un evento custom da un interceptor o dal backend.

Volendo uniformare gli eventi tra upload e download e astrarci dalla libreria HTTP, aggiungiamo una nostra enumerazione e una nostra interfaccia al progetto:

export interface ProgressStatus {

  status: ProgressStatusEnum;

  percentage?: number;

}

export enum ProgressStatusEnum {

  START, COMPLETE, IN\_PROGRESS, ERROR

}

Nel componente potete vederne l'utilizzo: semplicemente emettiamo lo START all'avvio dell'operazione, IN\_PROGRESS in corrispondenza di UploadProgress, COMPLETE in corrispondenza di Respose ed ERROR in caso di errore.

Il template prevede una input file nascosta e un pulsante “UPLOAD”. Cliccando sul pulsante, si aprirà la finestra di selezione della input, dando modo all’utente di selezionare il file. Sul change della input richiamiamo il metodo upload del componente.

<div class="container-upload">

    <button [disabled]="disabled" [ngClass]="{'disabled': disabled}" class="button upload" (click)="inputFile.click()">

        UPLOAD

    </button>

    <input name="file" id="file"(change)="upload($event)" type="file" #inputFile hidden>

</div>

Al termine dell’operazione, sia in caso di successo che di errore, va ripulita la input dalla selezione, altrimenti risulta impossibile effettuare l’upload dello stesso file consecutivamente. Per fare questo, si può utilizzare sia *ReactiveForm* associando un *FormControl* alla input, sia con *ngModel* andando a pulire la proprietà bindata.

Se non volete tirare dentro la gestione delle form per così poco, accettando di sporcare leggermente la logica del componente, potete utilizzare il decoratore *ViewChild*, per ottenere un riferimento all’elemento del DOM, tramite il quale potete resettare il valore dal *nativeElement*.

Per il download, facciamo un ragionamento simile, ma qui introduciamo un ulteriore requisito: vogliamo scaricare il file, aggiornare il progress e fornire il file scaricato all'utente senza ulteriore interazione.

Il markup del componente Download è molto semplice:

<button

 [disabled]="disabled"

 class="button download"

 [ngClass]="{'disabled': disabled}"

 (click)="download()">download</button>

La logica del componente è molto simile a quella dell'upload ma, come abbiamo detto, una volta terminato il download vogliamo subito fornire il file all'utente.

import { Component, Input, Output, EventEmitter } from '@angular/core';

import { HttpEventType } from '@angular/common/http';

import { UploadDownloadService } from 'src/app/services/upload-download.service';

import { ProgressStatus, ProgressStatusEnum } from 'src/app/models/progress-status.model';

@Component({

  selector: 'app-download',

  templateUrl: 'download.component.html'

})

export class DownloadComponent {

  @Input() public disabled: boolean;

  @Input() public fileName: string;

  @Output() public downloadStatus: EventEmitter<progressstatus>;

  constructor(private service: UploadDownloadService) {

    this.downloadStatus = new EventEmitter<progressstatus>();

  }

  public download() {

    this.downloadStatus.emit( {status: ProgressStatusEnum.START});

    this.service.downloadFile(this.fileName).subscribe(

      data => {

        switch (data.type) {

          case HttpEventType.DownloadProgress:

            this.downloadStatus.emit( {status: ProgressStatusEnum.IN\_PROGRESS, percentage: Math.round((data.loaded / data.total) \* 100)});

            break;

          case HttpEventType.Response:

            this.downloadStatus.emit( {status: ProgressStatusEnum.COMPLETE});

            const downloadedFile = new Blob([data.body], { type: data.body.type });

            const a = document.createElement('a');

            a.setAttribute('style', 'display:none;');

            document.body.appendChild(a);

            a.download = this.fileName;

            a.href = URL.createObjectURL(downloadedFile);

            a.target = '\_blank';

            a.click();

            document.body.removeChild(a);

            break;

        }

      },

      error => {

        this.downloadStatus.emit( {status: ProgressStatusEnum.ERROR});

      }

    );

  }

}

</progressstatus></progressstatus>

Per farlo dobbiamo leggermente sporcarci le mani, manipolando il DOM dal componente. Questo perché il modo più veloce di scaricare il file, senza un' ulteriore interazione dell'utente, è creare al volo un elemento *anchor* e agganciarci un object URL costruito a partire dal *blob* scaricato. Possiamo poi impostare la proprietà download dell'elemento con il nome del file e, a quel punto, cliccando l'elemento da codice, riusciamo a ottenere un download diretto anche per i file che il browser sa aprire, come i PDF. Subito dopo possiamo eliminare l'*anchor* creato.

Un po' bruttino, concordo. Probabilmente la cosa migliore sarebbe spostare tutta la parte di manipolazione in una direttiva, ma ai fini del nostro ragionamento non cambierebbe molto: ve lo lascio come esercizio!

Utilizziamo entrambi i componenti dal componente FileManager e il gioco è fatto. L'HTML sarà il seguente:

<app-upload [disabled]="showProgress" (uploadStatus)="uploadStatus($event)"></app-upload>

<h2>File List</h2>

<p \*ngIf="showProgress"> progress <strong>{{percentage}}%</strong></p>

<hr>

<div class="container">

    <ul>

        <li \*ngFor="let file of files">

            <a>

                {{file}}

                <app-download [disabled]="showProgress" [fileName]="file" (downloadStatus)="downloadStatus($event)"></app-download>

            </a>

        </li>

    </ul>

</div>

La logica del componente  si limiterà a recuperare la lista dei file disponibili e a mostrare lo stato di upload e download sulla base degli eventi ricevuti dai componenti figli:

import { Component, OnInit } from '@angular/core';

import { UploadDownloadService } from 'src/app/services/upload-download.service';

import { ProgressStatusEnum, ProgressStatus } from 'src/app/models/progress-status.model';

@Component({

  selector: 'app-filemanager',

  templateUrl: './file-manager.component.html'

})

export class FileManagerComponent implements OnInit {

  public files: string[];

  public fileInDownload: string;

  public percentage: number;

  public showProgress: boolean;

  public showDownloadError: boolean;

  public showUploadError: boolean;

  constructor(private service: UploadDownloadService) { }

  ngOnInit() {

    this.getFiles();

  }

  private getFiles() {

    this.service.getFiles().subscribe(

      data => {

        this.files = data;

      }

    );

  }

  public downloadStatus(event: ProgressStatus) {

    switch (event.status) {

      case ProgressStatusEnum.START:

        this.showDownloadError = false;

        break;

      case ProgressStatusEnum.IN\_PROGRESS:

        this.showProgress = true;

        this.percentage = event.percentage;

        break;

      case ProgressStatusEnum.COMPLETE:

        this.showProgress = false;

        break;

      case ProgressStatusEnum.ERROR:

        this.showProgress = false;

        this.showDownloadError = true;

        break;

    }

  }

  public uploadStatus(event: ProgressStatus) {

    switch (event.status) {

      case ProgressStatusEnum.START:

        this.showUploadError = false;

        break;

      case ProgressStatusEnum.IN\_PROGRESS:

        this.showProgress = true;

        this.percentage = event.percentage;

        break;

      case ProgressStatusEnum.COMPLETE:

        this.showProgress = false;

        this.getFiles();

        break;

      case ProgressStatusEnum.ERROR:

        this.showProgress = false;

        this.showUploadError = true;

        break;

    }

  }

}

Potete scaricare i sorgenti dal mio GitHub al seguente indirizzo:

<https://github.com/AARNOLD87/down-up-load-with-angular>