Sistemes i Tecnologies Web **Curs 2020-2021**

Pràctica: *WebArchiver*

Índex

1 Objectius

Els objectius d'aquesta pràctica són els següents:

- Consolidar els conceptes de JavaScript que hem vist a teoria.
- Treballar amb els mòduls que ens ofereix NodeJS.
- Implementar amb JavaScript i NodeJS el *client-side* i el *server-side* d'una aplicació web.

2 Enunciat

Desenvoluparem una aplicació web per a descarregar una URL en un fitxer comprimit en format zip. Aquesta descàrrega serà recursiva, cosa que implica que, si el destí d'una URL inclou enllaços, ja siguin relatius a la pròpia aplicació web, com absoluts, aquests enllaços també s'hauran de descarregar i comprimir en el fitxer zip resultant. Finalment, aquest fitxer zip l'enviarem al client. **Atenció: el servidor, en cap moment, guarda a disc cap fitxer!**

El nivell de recursivitat de descàrrega d'enllaços el podrem configurar. Com que seria possible que una URL impliqués haver de descarregar-se moltíssims fitxers, també podrem configurar el nombre màxim de fitxers que volem descarregar-nos. Superat aquest llindar, la resta de fitxers de la descàrrega no s'afegirien al fitxer zip.

3 Funcionalitat del front-end

El front-end de la nostra aplicació web consistirà en un formulari, tal i com es mostra a la Figura 1, a través del qual podrem especificar la URL a descarregar, el nivell de recursivitat màxim i el nombre màxim de fitxers a descarregar.

Figura 1: Formulari a on especifiquen la URL i la configuració de la descàrrega.

4 Funcionalitat del back-end

Us explicarem la funcionalitat del back-end a través del següent cas d'ús, a on la URL que especifiquem al formulari de descàrrega és: `http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`.

El codi HTML d'aquesta pàgina és:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<body>
<h1>This is test</h1>
<ul>
<li><a href="test_1_1.html">test1_1.html</a></li>
<li><a href="test_1_2.html">test1_2.html</a></li>
<li><a href="test_1_1.html">test1_1.html</a></li>
</ul>
</body>
</html>
```

A la següent taula us mostrem els nivells de profunditat, pel que fa als enllaços, d'aquesta web:

Pàgina inicial (nivell 1)	**Segon nivell**	**Tercer nivell**
test.html	**test_1_1**.html	**test_2_1**.html
		test1.html
	test_1_2.html	**test_2_2**.html

4.1 Descàrrega recursiva d'una web en un fitxer zip

Recordeu que un fitxer zip és un *archive*. Un *archive* conté entrades. Cada un dels enllaços que ens anem descarregant generen una entrada en el *archive*.

Quan ens descarreguem la URL especificada en el formulari de descàrrega, sempre generarem una entrada al *archive* amb nom **index**.html. Si aquesta URL conté enllaços, cada un dels enllaços generaran una entrada en el *archive* amb el nom:
`contador.extensió_recurs_original`, p.e.: 1.html, 2.pdf, 3.html, 4.gz, etc...

Si en el formulari inicial especifiquem dos nivells de profunditat de descàrrega, el servidor ha de retornar un fitxer **zip** amb les següents entrades:

```
Archive:  files.zip
Length      Date    Time    Name
---------  ---------- -----   ----
307        2018-05-10 10:58   index.html
238        2018-05-10 10:58   1.html
210        2018-05-10 10:58   2.html
---------                     -------
755                           3 files
```

A l'exemple anterior, la correspondència entre les URL's descarregades i la seva corresponent entrada en el *zip archiver* seria la següent:

URL descarregada	zip archiver entry
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`	index.html
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_1.html`	1.html
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html`	2.html

El primer nivell de descàrrega és la URL especificada en el formulari de descàrrega. Entrem en la recursivitat quan, dins d'aquesta web trobem altres enllaços. Com que el nivell de recursivitat especificat al formulari és 2, els nivells de la descàrrega serien la primera i la segona columna que es mostra a la taula 4. Per tant els enllaços que continguessin les pàgines `test_1_1.html` i `test_1_2.html` no es descarregarien.

La descàrrega s'ha de fer directament sense passar per disc, utilitzant *streams* i *pipes* entre *streams*.

A mesura que anem descarregant una web, a través d'un *inputStream*, anem afegint el seu contingut en un stream, de tipus *transformStream*, que parsejarà la web que estem descarregant tot cercant tags HTML de tipus `<a>`.

A mesura que les dades van passant a través del *transformStream* les anem enviant a un *outputStream* de tipus zip.

La següent figura mostra la relació, l'encadenament, dels streams:

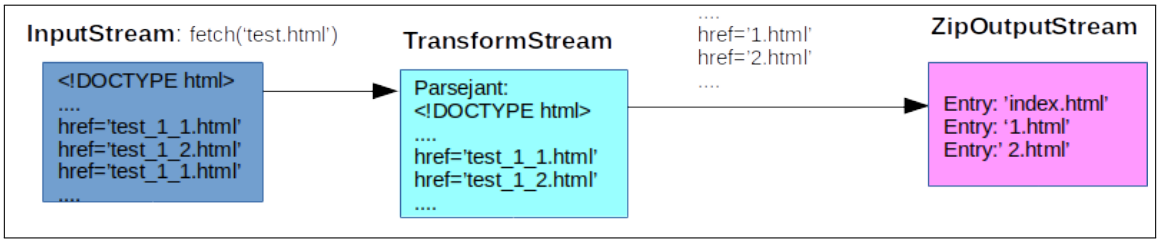

Figura 2: Encadenament d'streams.

Quan el servidor pot confirmar que ja s'han descarregat tots els fitxers, en funció del nivell de recursivitat especificat, tanca el *outpuStream*, de tipus zip i ho envia al client. Aquesta és la resposta a la petició HTTP que ha rebut del client quan hem fet el submit del formulari de descàrrega.

Noteu que en aquest procès, quan estem parsejant el contingut d'una web, a través del *transformStream*, si trobem un tag de tipus `<a>` hem d'iniciar un nou procés de descàrrega, a través de *streams*, i connectar la descàrrega mitjançant un *pipe*, amb el *outputStream* de tipus zip que estem construint. A la figura 4 podeu veure un esquema d'aquest procès recursiu.

Vegeu com treballar amb *streams* i *pipes* en el document de conceptes bàsics de NodeJS.

4.2 Traducció dels noms dels enllaços dins de les pàgines html que ens descarreguem

Com hem vist, a la secció anterior, i en funció del cas d'ús amb el que estem treballant, si al formulari de descàrrega especifiquéssim tres nivells de recursivitat, el servidor ens tornaria el següent fitxer zip:

```
Archive:  files.zip
Length      Date        Time    Name
---------  ----------  -----   ----
307         2018-05-11 09:13   index.html
238         2018-05-11 09:13   1.html
210         2018-05-11 09:13   2.html
148         2018-05-11 09:13   3.html
145         2018-05-11 09:13   4.html
148         2018-05-11 09:13   5.html
---------                      -------
1196                           6 files
```

La relació entre les entrades d'aquest zip i les URL's descarregades seria la següent:

Pàgina inicial (nivell 1)	Segon nivell	Tercer nivell	Entrada en el archiver
test.html			index.html
	test_1_1.html		1.html
		test_2_1.html	4.html
		test1.html	5.html
	test_1_2.html		2.html
		test_2_2.html	3.html

Si descomprimim el fixer zip i al navegador obrim el fitxer **index.html**, volem navegar normalment per aquesta pàgina accedint als enllaços que també ens hem descarregat. Ara bé, els enllaços ens els hem descarregat amb nom: `[1.html .. 5.html]`. Aquest canvi de nom s'ha de reflectir a les pàgines web a on hi han aquests enllaços. Per tant, quan la pàgina que ens estiguem descarregant contingui enllaços, ja siguin relatius o bé absoluts, els traduïrem: **canviarem l'html de la pàgina** per a que faci referència als arxius obtinguts en descomprimir el zip:
`[1.html ... 5.html]`.

Així doncs, el codi del fitxer `index.html` descarregat seria el següent:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<body>
<h1>This is test</h1>
<ul>
<li><a href="1.html">test1_1.html</a></li>
<li><a href="2.html">test1_2.html</a></li>
<li><a href="1.html">test1_1.html</a></li>
</ul>
</body>
</html>
```

Noteu com ha canviat el valor de l'atribut **href** dels tags **<a>** en relació a la pàgina `test.html` original.

4.2.1 Enllaços que ja ens hem descarregat

En aquest cas d'us que us plantegem, noteu que tot i que la pàgina que estem descarregant, `test.html`, conté tres enllaços, només ens hem descarregat els fitxers `1.html` i `2.html`. Això és així ja que la pàgina *test.html* conté dos enllaços iguals. Per tant, un cop hem descarregat un d'ells l'altre no s'ha de descarregar. Així només hem de generar una única entrada en el *zip archiver*.

4.3 Descàrrega d'un nombre màxim de fitxers

Fóra possible que la descàrrega d'un lloc web impliqués la descàrrega de molts fitxers, cosa que implicaria moltes entrades en el *zip archiver*. Mitjançant un camp del formulari de descàrrega, especificarem el nombre màxim de fitxers que ens volem descarregar.
En cas que, en els processos de parseig de les URL's que ens estem descarregant, assolíssim el nombre màxim de fitxers a descarregar, els següents enllaços trobats en aquest procès de parseig, no els descarregaríem. Aquests enllaços no descarregats els traduïríem amb el nom '**404.html**'.
El fitxer **404.html** no existeix, ni en local, ni l'hem de descarregar d'enlloc, per tant no l'inclourem com a entrada en el *archiver*. En la pàgina descarregada resultaria com un enllaç trencat.
En el cas d'us que estem utilitzant, si en el formulari de descàrrega, en el camp **maxFiles**, especifiqueu 2 fitxers, el fitxer zip que us descarregareu tindrà les següents entrades:

```
Archive:  files.zip
Length      Date      Time    Name
---------  ---------- -----   ----
307        2018-05-10 10:58   index.html
238        2018-05-10 10:58   1.html
---------                     -------
545                           2 files
```

Noteu que no ens hem descarregat el fitxer corresponent a l'enllaç:

http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html

El contingut de la pàgina **index.html** que us descarregueu serà:

```
<li><a href="1.html">test1_1.html</a></li>
<li><a href="404.html">test1_2.html</a></li>
<li><a href="1.html">test1_1.html</a></li>
....
```

4.4 Gestió de URL's que no existeixen

Per a facilitar la pràctica no considerarem aquest cas.

5 Estructura de directoris de la pràctica

A continuació us comentem l'estructura de directoris i els fitxers que ens caldran per implementar l'aplicació.

5.1 webArchiver/

Directori d'on penjaran tots els fonts i subdirectoris de l'aplicació.

5.2 URLManger.js

Fitxer que conté la classe URLManager.

5.3 ReplaceManger.js

Fitxer que conté la classe ReplaceManager.

5.4 apps.js

És el fitxer principal de la nostra aplicació l'executarem amb la comanda:

```
cd webArchiver
node app.js
```

A continuació us indiquem, a grans trets, la funcionalitat bàsica que ha d'implementar el servidor, i per tant, que implementarem en aquest fitxer:

1. En primer lloc, heu de crear un servidor HTTP, posar-lo a escoltar peticions en el port **3000**. Per a fer-ho utilitzarem el framework Express. Vegeu més informació a [7] i als apunts de teoria de node.
2. Quan us arribi una petició HTTP heu de gestionar la petició per detectar el recurs que us estan demanant i tornar-ho al client. Aquest routing també el farem mitjançant Express. Vegeu com fer-ho a [8] i als apunts de teoria de node.
3. Quan la petició HTTP que rebeu sigui la de descàrrega d'una URL, heu d'iniciar el procès de descàrrega recursiu de la URL, enllaçant-ho amb la creació del fitxer `.zip`. Aquest fitxer `.zip` és el que retornareu com a la resposta a la petició HTTP que us acaben de fer.

5.5 package.json

Com el seu nom indica, aquest fitxer és un JSON que té el nom de tots els mòduls de NodeJS que necessita l'aplicació i la seva versió.
Inicialment, aquest fitxer es genera amb la comanda: `npm init`. Cada cop que instal.leu un mòdul amb l'`npm` utilitzant el paràmetre `--save`, s'afegeix l'entrada del mòdul i de la seva versió a aquest fitxer. L'esquelet de la pràctica ja us proporciona el fitxer package.json amb els mòduls de node que calen. Per instal.lar-los heu d'executar la comanda:

```
npm install
```

Mireu la secció 9.1 per a més detalls sobre la utilització de la comanda `npm`.

5.6 public

Si us calen fitxers `.css` o llibreries `.js`, els podeu desar en aquest directori, en els subdirectoris, `css` i `js`, respectivament. Vegeu com servir aquests fitxers en la secció 6.

5.7 public/index.html

Contindrà el formulari de descàrrega d'una URL a través del qual l'usuari podrà especificar la URL, el nivell màxim de recursivitat i el nombre màxim de fitxers a descarregar.

5.8 test

Contindrà tots els tests unitaris.

5.9 node_modules

En aquest directori s'instal.laran de cop tots els mòduls de node especificats en el fitxer **package.json**. Trobareu un subdirectori per a cada mòdul.

6 Routing de les peticions HTTP que rep el servidor

El fitxer **app.js** defineix quins són els punts d'entrada (*end points*) de l'aplicació web. En el server anomenem *routing* al procés per gestionar cada punt d'entrada de l'aplicació i tornar la corresponent resposta al client. Les peticions HTTP que ha de gestionar el servidor, és a dir el *routing*, l'implementarem amb l'objecte **application** [10] de l'API d'Express. Teniu exemples de com fer-ho als apunts de node de teoria, a la secció de *basic-routing* de la web de *Express* [8] i en la secció de *routing* de la web de *Express* [9].

En primer lloc, us caldrà configurar un *middleware* (**app.use**) per servir directament els fitxers o subdirectoris dins del directori **public**. Aquests fitxers seran, bàsicament, llibreries de JavaScript i fulls d'estil. Això ho fareu amb la instrucció:

```
app.use(express.static(path.join(__dirname, 'public')));
```

Així, si us cal fer referència als fitxers estàtics que teniu al directori *public*, només heu de fer referència a la seva ubicació relativa al directori **public**. Per exemple, per a fer referència en el vostre codi html al fitxer **public/css/style.css**, ho faríeu de la següent manera:

```
<link rel="stylesheet" href="css/style.css">
```

Els punts d'entrada del servidor que haureu de gestionar seran els següents:

GET / Quan rebeu aquesta petició (GET / HTTP/2.0) heu de servir el fitxer **index.html**. Com que és un fitxer estàtic, si el deseu en el directori **public** de l'aplicació, gràcies a la utilització del *middleware d'Express* comentat anteriorment (**express.static(..."public")**) el servidor retornarà directament la pàgina **index.html**. En resum, no heu de programar res específic.

GET /crawler Quan rebeu aquesta petició **(GET /crawler?uri=...&maxFiles=...)** Heu d'iniciar el procés de descàrrega de la URL que us passen com a paràmetre **uri** de la petició GET.

Totes les peticions que li arribaran al servidor seran de tipus **get**, per tant per gestionar-les utilitzareu el mètode **app.get(path, callback [, callback ...])** de l'objecte **app** d'*Express*. Vegeu com funciona a [12].

Per obtenir els paràmetres de la *query string* utilitzareu el camp **query** de l'objecte **request**. El camp **query** és un objecte que conté com a propietats els paràmetres de la *query string*. Vegeu com funciona a [11].

Per enviar les capçaleres HTTP al client per indicar-li quin tipus de recurs li voleu enviar, ho podeu fer utilitzant la funció **response.writeHead(statusCode[, statusMessage][, headers])**. Vegeu la seva utilització a [4].

7 Disseny

A continuació us fem una proposta del disseny de l'aplicació. Aquest disseny contindrà funcions i classes. En el següent diagrama us mostrem, marcats amb una **C**, les classes, i amb una **F**, les funcions:

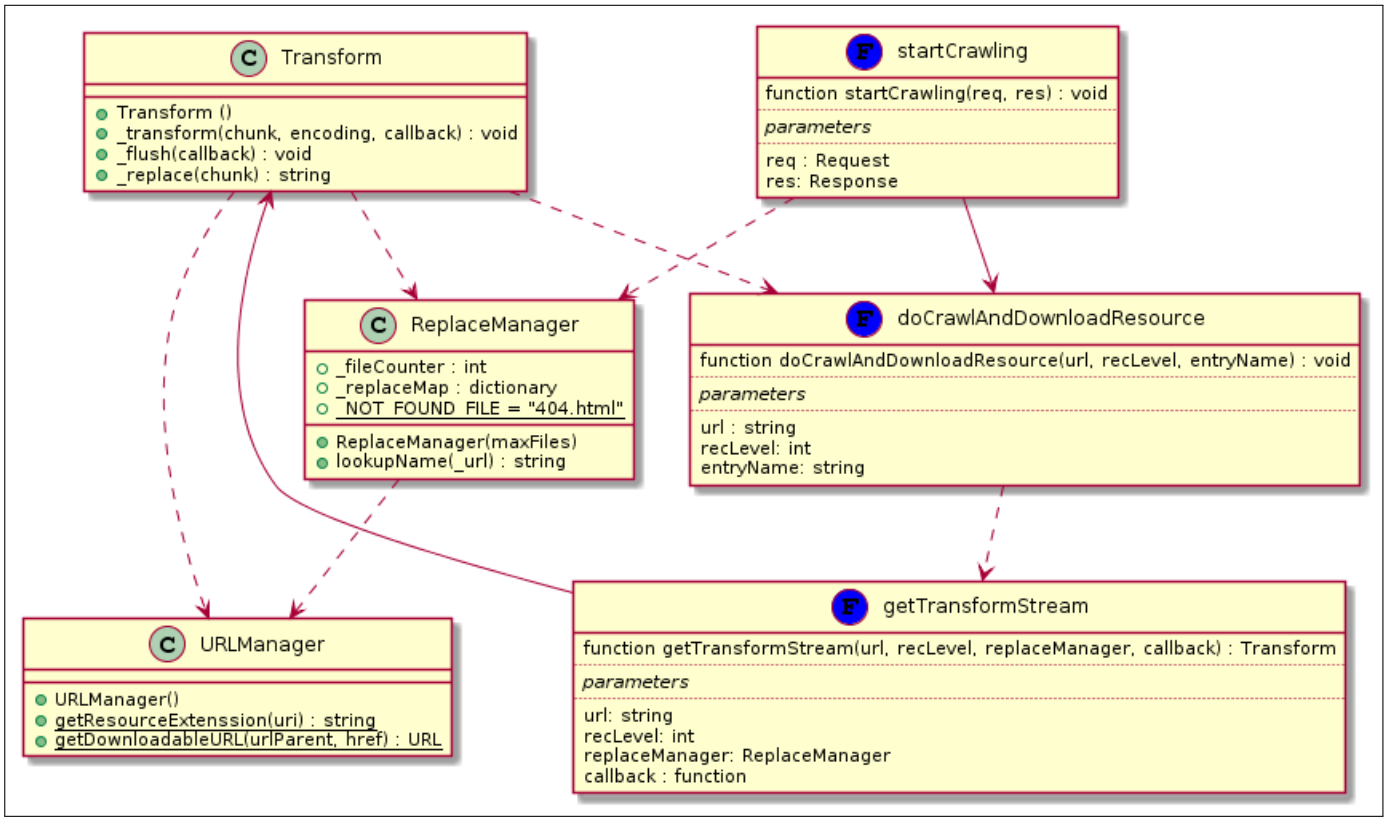

Figura 3: Diagrama de classes i funcions.

En el disseny, el que apareix com a subratllat, és estàtic.

A continuació us expliquem què fa cada funció i cada classe.

7.1 function URLManager()

Classe que encapsula mètodes per treballar amb URL's.
Els mètodes d'aquesta classe seran **tots estàtics**.

Constructor

`function URLManager()`

Mètodes

`{static} getResourceExtension(uri)`

Donada una URI obté l'extensió del recurs identificat per la URI incloent un punt. En cas que la URI no identifiqui explícitament un recurs concret retorna la cadena `".html"`.
Exemples:

url	return value
http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html	".html"
http://www.google.es	".html"
http://deic-docencia.uab.cat/test/foo?a=1&b=2	".html"
http://donotexist	".html"
http://foo.com/a.pdf	".pdf"
http://foo.com/a	".html"

parameters:

uri - Objecte de tipus URL que encapsula la URL d'on volem extraure l'extensió del recurs que identifica.

En primer lloc us caldrà obtenir el *path* de l'objecte URL passat com a paràmetre. Per a fer-ho heu d'accedir al camp **pathname** de l'objecte URL. Vegeu com fer-ho a [3]. Per extreure l'extensió del recurs identificat amb un *path* us caldrà utilitzar la funció: `path.extname(path)`. Vegeu com funciona a [5].

`{static} getDownloadableURL(urlParent, href)`

Mètode que **retorna un objecte URL** que encapsula una URL ben formada seguint el següent criteri: donada una URL absoluta i el valor de l'atribut **href** d'un tag <a>, si el paràmetre `href` és una url absoluta, retorna aquest valor directament. Si el paràmetre `href` és una url relativa, construeix la URL absoluta associada, combinant la URL absoluta del paràmetre `urlParent` i el paràmetre `href`.
parametres:

urlParent - URL absoluta de la pàgina que estem crawlejant.

href - Valor de l'atribut **href** d'un tag <a> que hem trobat en el procés de *crawlejar* la web especificada en `urlParent`.

En cas que el tag <a> no tingui l'atribut **href**, el valor d'aquest paràmetre serà **undefined**. Si es dona aquest cas, cal que tracteu el paràmetre **href** com una cadena buida.

Exemples:

urlParent:	http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html
href:	test_1_1.html
returned value:	http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_1.html
urlParent:	http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html
href:	http://www.google.es
returned value:	http://www.google.es
urlParent:	http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html
href:	""
returned value:	http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html

Aquesta funcionalitat la implementareu creant un objecte de la classe URL cridant al constructor `URL(input[, base])`. Vegeu com s'utilitza a [3].

Per a poder utilitzar aquesta classe des d'un altre fitxer, el que farem serà exportar-la tot utilizant el pattern *reveal module pattern*:
En el fitxer **`URLManager.js`**:

```
//Exportar els mètodes que ens calguin fer visibles.
//En aquest cas tota la classe.
module.exports={
        URLManager
}
```

Importeu la classe quan us calgui:

```
const URLManager = require('./URLManager').URLManager;
```

Ara ja podeu utilitzar-la normalment: `URLManager.getDownloadableURL(...)`

7.2 function ReplaceManager(maxFiles)

Aquesta classe encapsula un diccionari a on guardem la relació entre les URL's dels enllaços que trobem en parsejar una web, i el nom pels que els traduirem. Tots els índexs d'aquest diccionari seran URL's absolutes, i com a valor d'aquest index guardarem el nom de l'entrada en el *archiver* associada a la descàrrega d'aquest enllaç. Vegeu més informació a les seccions 4 i 4.1.
En el cas d'ús plantejat en la secció de funcionalitat, a on la pàgina **test.html** conté tres enllaços relatius al site:

```
...
<li><a href="test_1_1.html">test1_1.html</a></li>
<li><a href="test_1_2.html">test1_2.html</a></li>
<li><a href="test_1_1.html">test1_1.html</a></li>
....
```

en el procés de parsejar aquesta pàgina tindríem el següent diccionari:

Key	Value
http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html	index.html
http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_1.html	1.html
http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html	2.html

El resultat de parsejar la pàgina seria el que us mostrem a continuació. Noteu com utilitzarem els *values* del diccionari en el codi HTML resultant. Aquest codi és el que enviarem a l'stream de compressió:

```
<li><a href="1.html">test1_1.html</a></li>
<li><a href="2.html">test1_2.html</a></li>
<li><a href="1.html">test1_1.html</a></li>
....
```

Noteu, també, com utilitzarem aquests *values* del diccionari en les entrades del fitxer zip que retornarem al navegador:

```
Archive:  files.zip
Length      Date    Time    Name
---------  ---------- -----   ----
307        2018-05-10 10:58   index.html
238        2018-05-10 10:58   1.html
210        2018-05-10 10:58   2.html
---------                     -------
755                           3 files
```

Constructor

function ReplaceManager(maxFiles)

parameters:

maxFiles Nombre màxim de fitxers que pot contenir el zip. Per tant, és el nombre màxim de URLs que ens podem descarregar. Aquest paràmetre el podeu tractar com un paràmetre normal d'una funció.

Camps

_fileCounter: Comptador. Utilitzarem aquest camp com a nom de l'entrada en el zip archiver.

_replaceMap: Diccionari a on guardem la relació entre les URL's que ens estem descarregant i el nom de l'entrada en el *archiver* per a aquesta URL.

_NOT_FOUND_FILE: Camp **estàtic** amb la cadena: "404.html".

Mètodes

`lookupName(_url)`

Mètode que chequeja si el paràmetre **`_url`** és un índex del diccionari. En cas positiu, retorna el valor d'aquest index. En cas negatiu afegeix aquesta `url`, com a índex del diccionari, tot associant-hi com a valor, el nom que s'utilitzarà a l'*archiver* com a entrada per a aquesta URL. Aquest nom és el valor actual de la propietat **`_fileCounter`** + l'extensió que té el recurs en la url. Per obtenir aquesta extensió us caldrà utilitzar el mètode **`URLManager::getResourceExtension(url)`**.
Recordeu que la propietat **`_fileCounter`** s'ha de incrementar cada cop que la utilitzeu com a nom.
Heu de tenir en compte les següents casuístiques:

- **Al primer enllaç** que registrarem en el diccionari, és a dir, quan el diccionari estigui buit, sempre li assignarem el valor **`index.html`**.
- Si en fer la crida a aquest mètode, la `url` no existeix en el diccionari, però hem superat el nombre màxim de fitxers a descarregar, especificat en el paràmetre **`maxFiles`**, afegim al diccionari aquesta nova `url` i com a valor li associem la cadena **`404.html`**.

En qualsevol cas, aquest mètode ha de retornar el nom del fitxer que s'utilitzarà com a entrada en el *archive*, i que, també utilitzarem com a nou valor de l'atribut **`href`** de les pàgines que ens estem descarregant.
Vegeu-ne el següent cas d'us a on fixem el paràmetre **`maxFiles`** a 2:

```
var r = new ReplaceManager(2);
console.log(r.lookupName('http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html'));
> index.html
console.log(r.lookupName('http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_1.html')));
> 1.html
console.log(r.lookupName('http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html')));
> 404.html
```

Noteu pel cas de l'enllaç: **`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html`**, com que hem superat el nombre màxim de fitxers a descarregar, el mètode ens torna la cadena **`"404.html"`**. Aquesta cadena apareixerà com a valor de l'atribut **`href`** per a aquest enllaç. Aquest enllaç no el descarregarem, per tant no hi haurà cap entrada associada a l'*archiver*.
En aquest cas d'us acabaríem tenint el següent diccionari:

Key	Value
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`	index.html
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_1.html`	1.html
`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html`	404.html

Noteu com utilitzarem aquests *values* del diccionari en el codi HTML que enviarem a l'stream de compressió:

```
<li><a href="1.html">test1_1.html</a></li>
<li><a href="404.html">test1_2.html</a></li>
<li><a href="1.html">test1_1.html</a></li>
....
```

Noteu també que no ens hem descarregat el fitxer corresponent a l'enllaç:

`http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test_1_2.html`

```
Archive:  files.zip
Length      Date    Time    Name
---------  ---------- -----   ----
307        2018-05-10 10:58   index.html
238        2018-05-10 10:58   1.html
---------                     -------
545                           2 files
```

Per a poder utilitzar aquesta classe des d'un altre fitxer, el que farem serà exportar-la tot utilizant el pattern *reveal module pattern*:
En el fitxer **`ReplaceManager.js`**:

```
//Exportar els mètodes que ens calguin fer visibles.
//En aquest cas tota la classe.
module.exports={
        ReplaceManager
}
```

Importeu la classe quan us calgui:

```
const ReplaceManager = require('./ReplaceManager').ReplaceManager;
```

Ara ja podeu utilitzar-la normalment:

```
let replaceManager = new ReplaceManager(...);
```

7.3 `function startCrawling(req, res)`

Aquesta funció la crida el servidor quan rep la petició HTTP del tipus:
`GET /crawler?uri="www.google.es"&maxFiles=2&recLevel=3`.
Tingeu en compte que la *query string* d'aquesta petició HTTP és:
`uri="www.google.es"&maxFiles=2&recLevel=3`.

Aquesta funció analitza els paràmetres de la *query string* per obtenir: la URL a descarregar, el nivell de recursivitat i el nombre màxim de fitxers a descarregar.
Per a accedir als paràmetres de la *query string* ho farem mitjançant l'objecte **`request`** del framework *Express*, accessible des del paràmetre **`req`** del mètode **`startCrawling`**. Concretament utilitzareu la propietat **`query`** d'aquest objecte. Aquesta propietat conté un diccionari amb tots els paràmetres de la *query string* i el seu valor. Vegeu com funciona a [6].
A continuació, ha d'invocar a la funció **`doCrawlAndDownloadResource`** (secció 7.4), per iniciar el procés de descàrrega. Noteu que a la funció **`doCrawlAndDownloadResource`**, cal que li passem el paràmetre **`entryName`**, que correspon al nom del fitxer que utilitzarem com a entrada en el `archiver`.
Per la primera URL a descarregar, la que obtenim del formulari, el nom del fitxer ha de ser **`index.html`**. Aquest nom l'obtenim invocant al mètode **`ReplaceManager::lookupName`**. Amb la crida a aquest mètode, s'afegirà una entrada al diccionari **`ReplaceManager::_replaceMap`**, tot associant la URL que obtenim del formulari (la URL incial) amb el nom **`index.html`** i retornant aquest nom: **`index.html`**.
En aquesta funció podeu definir un array (**`downloadedFiles`**) a on s'aniran guardant els noms dels fitxers que hem utilitzat com a entrades en l'`archiver` i que corresponen a les URL's que ja ens hem descarregat.

Paràmetres:

req Objecte que encapsula la petició HTTP que hem rebut per part del client.

res Objecte que usarem per enviar la resposta al client.

7.4 `function doCrawlAndDownloadResource(url, recLevel, entryName)`

Funció que es descarrega una URL, en parseja el seu contingut transformant el valor dels atributs **`href`** i envia la transformació al fitxer `.zip`. Aquest fitxer zip s'envia, com a resposta, al client. A aquesta funció li caldrà accedir per clausura a algunes variables definides en la funció **`startCrawling`** 7.3.

Paràmetres:

url URL que ens volem descarregar expressada amb el format:

```
URL = protocol:[//domain[:port]]path[?query][#fragment]
```

recLevel Sencer que indica el nivell de recursivitat de la descàrrega.

entryName Nom del fitxer que figurarà en l'entrada del archiver (zip) per a aquesta URL.

Aquesta funció ha de decidir si es descarrega una URL, o no, en funció de si ja estem al nivell màxim de recursivitat, de si ja hem assolit el nombre màxim de URL's a descarregar, o bé de si la URL ja ens l'hem descarregat prèviament.
Quan decidiu descarregar-vos una URL, cal que afegiu el nom del fitxer associat a la URL, corresponent al paràmetre **`entryName`**, en l'array que utilitzem per mantenir el control dels fitxers que ja ens hem descarregat.

Utilitzarem la funció **`fetch`** del mòdul **`node-fetch`** de nodeJS per fer una petició HTTP i obtenir un *inputStream* per descarregar-nos la URL. Vegeu la seva utilització a la següent secció 7.4.1.

Un cop haguem creat un *inputStream* associat a la URL a descarregar, l'heu de concatenar amb l'*stream* de transformació, que parseja la URL que ens estem descarregant.

Com sabeu, en el zip archiver anem afegint entrades. Cada entrada és un ***Stream*** d'on obtenim les dades a comprimir.

Un cop hem afegit totes les *entries* al *zip archiver*, aquest s'ha de 'tancar' executant el mètode **`zip.finalize()`**.
Un cop tancat el *zip archiver*, si l'heu concatenat amb algun *outputStream*, les dades comprimides s'envien a aquest *outputStream*.
Us haureu de pensar un mecanisme per detectar si el *zip archiver* que esteu muntant ja es pot tancar o no. Plantegeu-vos quan heu d'invocar al mètode **`archiver::finalize()`**. Aquest mètode s'ha d'invocar quan ja no calgui afegir més entrades a l'*archiver*. Recordeu que les entrades a l'*archiver* s'afegeixen amb la crida al mètode **`archiver::append`**. També penseu que la crida al mètode **`archiver::append`** és asíncrona. És a dir, podem afegir una entrada en el archiver i anar rebent en el archiver, asíncronament, les dades de l'*stream* corresponent a aquesta entrada. Es pot donar el cas que invoqueu al mètode **`archiver::finalize()`** quan encara estigueu rebent dades dels streams que formen part d'entrades en el archiver. No hi ha cap problema. Lo únic que no podreu fer, un cop invocat el mètode **`archiver::finalize`**, és afegir una nova entrada en el archiver.
Penseu en el problema del **`asyncMap`**.

7.4.1 fetch

Aquesta funció està inclosa el el mòdul **`node-fetch`** de nodeJS. Aquesta funció permet que des de el servidor puguem fer peticions HTTP i rebre la resposta. Donat que és una funció asíncrona, està implementada utilitzant promeses. Vegeu com s'utilitza a [18].
A continuació us mostrem un exemple de la utilització de la funció **`fetch`** per a obtenir, asíncronament, un *stream* amb la resposta d'una petició HTTP llençada des del servidor:

```
fetch('http://foo.com')
.then(fetchRes => {
        console.log(fetchRes.body)
})
.catch(err => console.error(err));
```

Atenció! **`fetchRes.body`** és l'*stream* que conté la resposta de la petició HTTP.

7.5 `function getTransformStream(url, recLevel, replaceManager, doCrawlAndDownloadResource)`

Aquesta funció implementa el patró de disseny *revealing module pattern* per tornar, revelar, un objecte, un *stream* de tipus **`Transform`**[1].
Aquest stream de transformació el programarem per a que les dades que li entrin a l'stream, que seran les dades del *inputStream* que establim en fer la petició HTTP per descarregar una url, les parsegi tot cercant els tags **`<a>`**.
Cada cop que trobi un tag **`<a>`** n'obtidrà el valor de l'atribut **`href`**. A partir del valor de **`href`** farà el següent:

1. Obtindrà la URL absoluta associada al valor de l'atribut **`href`**.
2. Utilitzarà la classe **`ReplaceManager`** per obtenir el nou valor de l'atribut **`href`**, que serà el nom de l'entrada en el *archiver* per aquest enllaç (*1.html*, *2.html*, *3.pdf*).
3. Com que és un stream de **transformació** n'aplicarà una: Agafarà el valor de l'atribut **`href`** i el modificarà pel nom generat amb la classe **`ReplaceManager`**.

   En el cas d'ús amb el que treballem, a on el contingut de la la url:
   `http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`

   és:

   ```
   ...
   <li><a href="test_1_1.html">test1_1.html</a></li>
   <li><a href="test_1_2.html">test1_2.html</a></li>
   <li><a href="test_1_1.html">test1_1.html</a></li>
   ...
   ```

   La sortida del stream de transformació serà:

   ```
   ...
   <li><a href="1.html">test1_1.html</a></li>
   <li><a href="2.html">test1_2.html</a></li>
   <li><a href="1.html">test1_1.html</a></li>
   ...
   ```

4. Iniciarà el procés per descarregar aquesta nova url absoluta construïda a partir del valor de l'atribut **`href`**.

La següent figura mostra com es relacionen els streams:

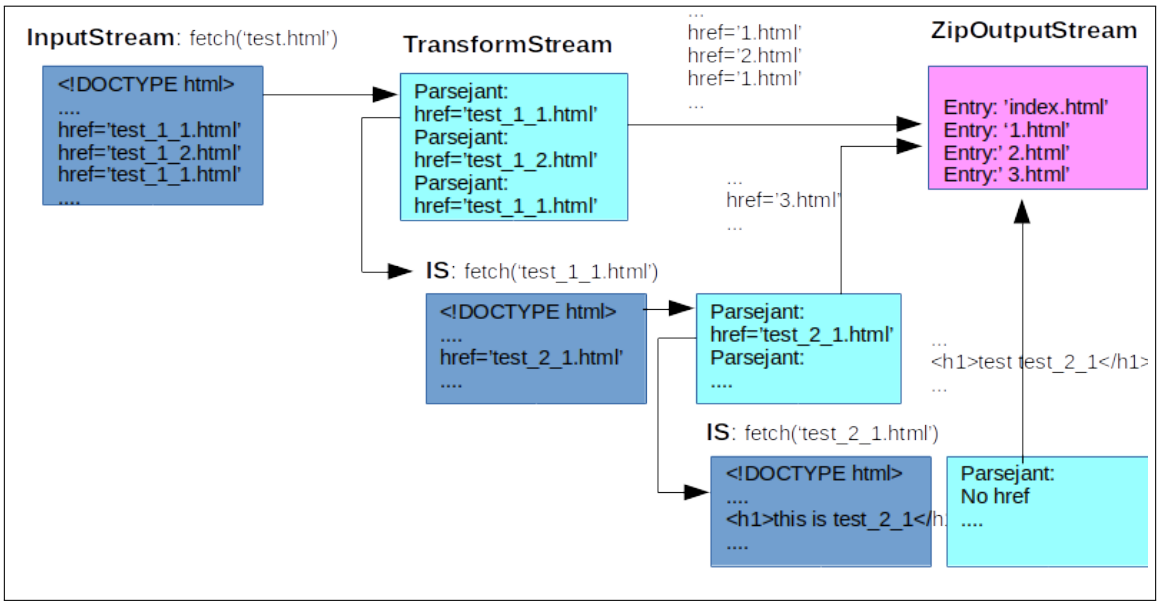

Figura 4: Exemple d'utilització dels streams en la descàrrega de la web `test.html`

[1] En el document de conceptes bàsics teniu informació de com crear streams de transformació i detalls del patró de diseny *revealing module pattern*.

Paràmetres:

url URL que estem parsejant expressada amb el format:

```
URL = protocol:[//domain[:port]]path[?query][#fragment]
```

recLevel Sencer que indica el nivell de recursivitat de la descàrrega en el que ens trobem.

replaceManager L'objecte **`replaceManager`** que conté el diccionari que estem utilitzant per fer les traduccions dels noms dels enllaços.

doCrawlAndDownloadResource La funció que utilitzarem per descarregar-nos cada enllaç que trobem mentre estem fent el parseig del contingut de la URL passada com a paràmetre.

Mètodes privats a implementar utilitzant l'aproximació del *module pattern*:

Tot seguint el patró *module pattern*, heu de crear un objecte de tipus **`Transform`** i li heu de definir els següents mètodes privats:

`_transform(chunk, encoding, callback)`

Aquest mètode va concatenant els troços de dades, **`chunk`**, que entren a l'stream de transformació en una variable global a la funció: **`getTransformStream`**::**`buffer`**.
Després invoca a la funció de **`callback`**.
Aquest mètode ja us el donem fet en l'esquelet de la pràctica.

`_flush(callback)`

Aquest mètode es crida quan ja han entrat les darreres dades a parsejar en el stream de transformació.
En aquest moment ja sabem que tenim totes les dades, el contingut de la URL que hem demanat via HTTP, en la variable **`buffer`**.
En aquest moment invoquem al mètode: **`_replace`** que fa el parseig del contingut de la URL tot descarregant-se els enllaços interns, i traduïnt aquest enllaços pel nom de l'entrada en el *archiver*.
Aquest mètode ja us el donem fet en l'esquelet de la pràctica.

`_replace(chunk)`

Aquest mètode l'hem cridat des del mètode **`_flush`**, passant-li com a paràmetre **`chunk`** tot el **`buffer`** que conté el contingut de la URL a la que hem accedit via HTTP.
Aquest mètode utilitza el mòdul **`cheerio`** per trobar tots els tags **`<a>`** obtenir-ne el valor del seu atribut **`href`** i modificar aquest valor. Aquest codi ja us el donem fet.
Si voleu entendre com funciona el mòdul **`cheerio`** vegeu-ne més informació a [16].

8 Preguntes clau que us heu de formular i respondre

Quin serà el outputStream que utilitzarà el servidor per enviar el fitxer zip al client?

En quina variable teniu disponible l'*outputStream* que connecta directament el servidor amb el client?

Com creo les entrades del zip archiver i a on?

Com sabeu, la utilització mòdul del *zip archiver* es basa en afegir entrades en un objecte zip a través de la funció:
`zipObj.append(readStream, {name: "entry name"})`

Heu de pensar quin serà el valor de l'*readStream* d'aquesta funció. Quan tingueu la resposta a aquesta pregunta, també tindreu clar quan heu d'executar aquest **`append`**.

Quan dono per tancat el zip archiver?

És a dir, quan decidiu que ja no afegireu més entrades?

Com enllaço l'*inputStream* (que obtinc a través de la petició HTTP que estableixo per descarregar-me una URL) amb l'*stream* de transformació (que parseja la URL per trobar-hi enllaços) amb el *zipOutputStream* i amb l'*stream* que em connecta directament amb el client?

9 Appendix

9.1 npm: NodeJS Package Manager

NodeJS ve acompanyat d'una eina per a la gestió de móduls de NodeJS, o móduls JavaScript, en general, el **npm** [2]. És una eina molt útil i senzilla d'utilitzar per instal.lar els mòduls de JavaScript que volem utilitzar en la nostra aplicació de NodeJS.

A continuació us indiquem les comandes més freqüents:

- Instal.lar de forma local, és a dir en el directori de l'aplicacio, un módul:

  `npm install` *`modul`* `--save`

  Amb el paràmetre `--save` modifiquem el fitxer `package.json` tot afegint una entrada amb la versió del mòdul que acabem d'instal.lar.

  Si volguéssim instal.lar aquest módul on tenim instal.lat el NodeJS caldria que substituíssiu el paràmetre `--save` pel paràmetre `-g`, global. En aquest cas estaríem fent una instal.lació global del paquet, i per tant, el podríeu utilitzar des de qualsevol aplicació (no ho feu).

- Si al directori de l'aplicació teniu el fitxer `package.json` amb la llista de tots els mòduls que necessita l'aplicació i encara no els teniu instal.lats, els podeu instal.lar de cop amb la comanda:
  `npm install`
- Per veure quina és la versió d'algun mòdul que tenim instal.lat:
  `npm list -g | grep` *`modul`*
- Per veure quina és la darrera versió d'algun mòdul disponible en el repositori:
  `npm show` *`modul`* `version`

10 Com fer un deploy de la nostra aplicació

Fer un deploy de la nostra aplicació és molt fàcil. Només cal que us posicioneu en el directori `webArchiver` i executeu la comanda:

```
node app.js
```

Al directori de descàrregues usat pel navegador s'haurà descarregat el fitxer `files.zip`.

Per veure el contingut del fitxer podeu executar la comanda: `unzip -l files.zip`.
Per descomprimir el fitxer podeu utilitzar la comanda: `unzip files.zip`
Per finalitzar el deploy cal que feu un `Ctrl + C`.

11 Proposta de planificació de la pràctica per sessions

La pràctica és llarga, i la millor manera d'acabar-la amb èxit és completant una funcionalitat setmana a setmana. Així que us proposem una planificació de la pràctica on cada setmana hi haurà l'entrega, **opcional** però que conta com a nota, d'una part de la funcionalitat.

Planificació de la pràctica per sessions

A taula hi ha el resum de les dates de les sessions de pràctiques.

Sessió	Setmana	Funcionalitat
Sessió 1	22 Març	Desenvolupar el Routing del servidor i la funció **startCrawling**. Desenvolupar la funció **doCrawlAndDownload** per descarregar una URL, **sense recursivitat**, encapsulant-la en un fitxer `zip` amb entrada `index.html`.
Sessió 2	12 Abril	Programar les classes **`ReplaceManager`**, **`URLManager`**. Utilitzar la classe **`ReplaceManager`** per obtenir el nom de la *entry* del zip `index.html`. Utilitzar la funció **getTransformStream** per fer la traducció d'adreces de la URL incial **sense utilitzar recursivitat**.
Sessió 3	19 Abril	Descàrrega recursiva; Limitació del nivell de descàrrega.
-	**26 Abril**	Entrega de la pràctica una hora abans de la vostra sessió de laboratori.

A continuació, us proposem, de forma detallada, el treball previ i les entregues parcials que heu d'anar fent al llarg de les sessions:

11.1 Sessió 1: Setmana del 22 de Març

11.1.1 Treball previ que heu de portar fet abans de la sessió

Res.

11.1.2 Treball que farem durant la sessió i que haureu d'entregar la setmana següent

1. Desenvolupar el fitxer `index.html` amb el formulari de descàrrega.
2. Mitjançant el framework *Express* programar el routing del servidor.
3. Desenvolupar les funcions: **`startCrawling`**, secció 7.3, i **`doCrawlAndDownloadResource`**, secció 7.4, per a que siguin capaces de:
   (a) Després de fer un submit del formulari de descàrrega, el servidor rebrà la petició:

   `HTTP GET /crawler?uri=XX&recLevel=... HTTP/2.0`
   D'aquesta petició agafar el valor dels camps: `uri`, `recLevel` i `maxFiles`.
   Pel que fa la URL a descarregar, proveu amb:
   `http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`

(b) Des de la funció **doCrawlAndDownloadResource** descarregueu-vos aquesta URL sense comprimir-la ni traduir-la i envieu-la directament al navegador mitjançant pipes. El navegador l'ha de poder visualitzar sense problemes. Recordeu que el servidor ha de respondre amb la següent capçalera al client per indicar que li envia un fitxer html.

```
response.writeHead(200, {'Content-Type': 'text/html'})
```

Aquest serà el pas previ per a fer la funcionalitat següent:

(c) Modificar el codi anterior, perquè, en lloc d'enviar al client el contingut de la URL anterior, li envieu aquest contingut encapsulat en un *zip* que tingui com a entrada la cadena '**index.html**' (*hardcoded*). En aquest cas el servidor ha de respondre amb la següent capçalera al client per indicar que li envia un fitxer en format *zip*.

```
response.writeHead(200, {'Content-Type': 'application/zip',
'Content-Disposition' : 'attachment; filename=files.zip' });
```

11.2 Sessió 2: Setmana del 12 d'Abril

11.2.1 Treball previ que heu de portar fet abans de la sessió

Entrega del codi proposat en la sessió 1.
Aquest control està valorat amb **1 punt** de la nota final de la pràctica.

11.2.2 Treball que farem durant la sessió i que haureu d'entregar la setmana següent

- En aquesta sessió encara **no** hem de programar la descàrrega recursiva. Per deshabilitar-la, cal que comenteu la crida a la funció **doCrawlAndDownloadResource** que trobareu en el mètode **transformStream._replace**.
- Programar les classes **URLManager**, secció 7.1, i **ReplaceManager**, secció 7.2. Feu un test unitari per assegurar-vos de que funcionen perfectament. Podeu crear els fitxers **test/urlManagerUnitTest.js** i

  test/replaceManagerUnitTest.js que continguin els tests unitaris d'aquestes classes. Programeu primer el test unitari **urlManagerUnitTest.js**.
- Utilitzar el mètode **ReplaceManager::lookupName**, per a obtenir el nom de l'entrada de l'*archiver* per a la URL inicial. Aquest mètode us ha de retornar la cadena '**index.html**'.
- Utilitzar la funció **getTransformStream** per a obtenir l'stream que tradueixi tots els enllaços de la URL que ens estem descarregant, pels noms que hi hauran a les entrades del fitxer zip. **No implementeu encara la recursivitat**.
- Comprovar que la transformació ha funcionat . Executeu els següents tests utilitzant la URL de test:
  `http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`

  maxFiles=3 Us heu de descarregar la URL de test encapsulada en un fitxer zip. En descomprimir el fitxer zip, comprovar que la pàgina **index.html** conté els enllaços modificats: **1.html**, **2.html** i **1.html**.

  maxFiles=2 Us heu de descarregar la URL de test encapsulada en un fitxer zip. En descomprimir el fitxer zip, comprovar que la pàgina **index.html** conté els enllaços modificats: **1.html**, **404.html** i **1.html**.

11.3 Sessió 3: Setmana del 19 d'Abril

11.3.1 Treball previ que heu de portar fet abans de la sessió

Entrega del codi proposat en la sessió 2.
Aquest control està valorat amb **1 punt** de la nota final de la pràctica.

11.3.2 Treball que farem durant la sessió

- Programarem la descàrrega recursiva. En primer lloc us caldrà habilitar la crida a la funció:

  `doCrawlAndDownloadResource` en el mètode **`transformStream._replace`**.

- Plantegeu-vos quan heu d'invocar al mètode **`archiver::finalize()`**. Aquest mètode s'ha d'invocar quan ja no calgui afegir més entrades a l'*archiver*. Repasseu l'explicació de la funció **`doCrawlAndDownloadResource`** 7.4.

  Penseu en el problema del **`asyncMap`**.

- Comproveu que la descàrrega recursiva funciona descarregnat-vos la URL:
  `http://stw.deic-docencia.uab.cat/nodeJS/webArchiver/test.html`

  Descomprimiu el fitxer zip i assegureu-vos que esteu navegant per tot el site **en local**. També assegureu-vos que la traducció dels enllaços és correcta.

- Programar la limitació del nivell de recursivitat (*recLevel*).

- Programar la limitació del nombre màxim de fitxers a descarregar.

12 Detalls sobre l'entrega de l'exercici

La pràctica l'entregarem la setmana del **26 d'Abril**, una hora abans de la vostra sessió de pràctiques. Passada aquesta hora no heu fer més canvis en el repositori.

Referències

[1] **NodeJS website** URL: https://nodejs.org/en/

[2] **npm website** URL: https://www.npmjs.com/

[3] **node API for URL module.** URL: https://nodejs.org/dist/latest/docs/api/url.html#url_the_whatwg_url_api

[4] **Funció response.writeHead** URL: https://nodejs.org/dist/latest/docs/api/http.html#http_response_writehead_statuscode_statusmessage_headers

[5] **node API for path module.** URL: https://nodejs.org/dist/latest/docs/api/path.html#path_path_extname_path

[6] **Express API for request module.** URL: https://expressjs.com/en/5x/api.html#req.query

[7] **Express framework.** URL: https://expressjs.com/

[8] **Express basic routing.** URL: https://expressjs.com/en/starter/basic-routing.html

[9] **Express extended routing.** URL: https://expressjs.com/en/guide/routing.html

[10] **Objecte Application** URL: https://expressjs.com/en/5x/api.html#app

[11] **Objecte Request.query** URL: https://expressjs.com/en/5x/api.html#req.query

[12] **Funció app.get** URL: https://expressjs.com/en/5x/api.html#app.get

[13] **Funció response.sendFile** URL: https://expressjs.com/en/5x/api.html#res.sendFile

[14] **Implementing a transform stream.** URL: https://nodejs.org/dist/latest/docs/api/stream.html#stream_implementing_a_transform_stream

[15] **Exemple de la utilització de la funció fs.createWriteStream** URL: https://nodejs.org/dist/latest/docs/api/fs.html#fs_fs_createwritestream_path_options

[16] **npm cheerio package.** URL: https://www.npmjs.com/package/cheerio

[17] **zip archiver package from npm.** URL: https://www.npmjs.com/package/archiver

[18] **node-fetch module** URL: https://www.npmjs.com/package/node-fetch