Motors de plantilles: Nunjucks


Una **plantilla** és un document estàtic (típicament HTML, si parlem de documents web), en el qual s'intercalen o embeuen unes certes marques per a agregar una mica de dinamisme. Per exemple, podem deixar una estructura HTML feta amb un buit per a mostrar un llistat de notícies, i que aqueix llistat de notícies s'extraga d'una base de dades i s'afija a la plantilla dinàmicament, abans de mostrar-la.

1. Motors de plantilles

Existeixen diversos motors de plantilles que podem emprar en Express, i que faciliten i automatitzen el processament d'aquests fitxers i el reemplaçament de les corresponents marques pel contingut dinàmic a mostrar. Alguns exemples són:

- **Pug**, un motor bastant habitual en Express, amb una sintaxi específica basada en HAML, una abstracció del propi llenguatge HTML.
- **Mustache**, un motor que combina HTML amb codi JavaScript embegut amb una certa sintaxi especial. A partir d'aquest motor, s'han creat uns altres molt similars, com *HBS*, també conegut com Handlebars.
- **EJS**, sigles d'Effective JavaScript templating, un motor de plantilles bastant senzill d'utilitzar i integrar amb contingut HTML.
- **Nunjucks**, un motor de plantilles desenvolupat per l'equip de Mozilla, que també s'integra perfectament en Express.
- etc.

1.1. Instal·lació del motor de plantilles

Una vegada hàgem triat el nostre motor de plantilles, l'instal·larem en la nostra aplicació com un mòdul més de NPM, i l'enllaçarem amb Express a través del mètode `app.set`, com una propietat de l'aplicació. En aquestes anotacions farem ús del motor **Nunjucks**, un motor de plantilles desenvolupat per Mozilla, molt similar a *Handlebars* en la seua sintaxi, i adaptat al seu ús amb Express. Podeu consultar més informació en la seua web oficial.

El primer que farem serà descarregar-ho i instal·lar-ho en el nostre projecte amb el seu corresponent comando (des de la carpeta del projecte Node):

```
npm install nunjucks
```

Ho haurem d'incloure en l'aplicació principal, juntament amb la resta de mòduls necessaris...

```
const express = require('express');
const nunjucks = require('nunjucks');
...
```

Després, ho establim com a motor de plantilles en l'arxiu principal de la nostra aplicació, una vegada inicialitzada l'aplicació Express:

```
let app = express();
...
app.set('view engine', 'njk');
```

Finalment, també és necessari establir uns paràmetres de configuració del motor de plantilles, emprant per a això el seu mètode `configure`. En concret, establirem que acte-escape els caràcters que es mostren (per a evitar atacs per codi embegut, per exemple), i li indicarem quin objecte conté l'aplicació Express. A més, indicarem que les diferents plantilles o vistes les situarem en la subcarpeta `views` de l'aplicació:

```
nunjucks.configure('views', {
    autoescape: true,
    express: app
});
```

1.2. Ubicació de les plantilles

Per defecte, en una aplicació Express les plantilles s'emmagatzemen en una subcarpeta anomenada `views` dins del projecte Node. Així ho hem configurat amb la instrucció anterior `configure` per a Nunjucks. En el nostre cas, en haver triat aquest motor, aquestes plantilles tindran extensió `.njk`, com per exemple `index.njk`.

2. Primers passos amb Nunjucks

Definirem algunes plantilles amb Nunjucks i comprovar com mostren la informació dinàmica, i com se'ls pot passar aquesta informació des dels encaminadors.

2.1. Preparant el servidor principal

Per a això, ens basarem en el nostre exemple de contactes que hem vingut desenvolupant en sessions prèvies. Podem copiar la carpeta del projecte "*ContactesREST_v2*" de sessions anteriors (la que vam fer amb tota l'estructura de carpetes per a models i encaminadors), i canviar-la de nom a "*ContactesWeb*". Dins instal·lem Nunjucks i ho deixem configurat en l'aplicació principal com a motor de plantilles per a l'aplicació. També

podem instal·lar i configurar Bootstrap si volem, per a poder aplicar els seus estils. L'arxiu principal `index.js` quedarà més o menys així:

```
// Llibreries
const express = require('express');
const mongoose = require('mongoose');
const nunjucks = require('nunjucks');

// Encaminadors
const mascotes = require(__dirname + '/routes/mascotes');
const restaurants = require(__dirname + '/routes/restaurants');
const contactes = require(__dirname + '/routes/contactes');

// Connexió amb la BD
mongoose.connect('mongodb://127.0.0.1:27017/contactes');

// Servidor Express
let app = express();

// Configurem motor Nunjucks
nunjucks.configure('views', {
    autoescape: true,
    express: app
});

// Assignació del motor de plantilles
app.set('view engine', 'njk');

// Middleware per a peticions POST i PUT
// Middleware per a estils Bootstrap
// Encaminadors per a cada grup de rutes
app.use(express.json());
app.use(express.static(__dirname + '/node_modules/bootstrap/dist'));
app.use('/mascotes', mascotes);
app.use('/restaurants', restaurants);
app.use('/contactes', contactes);

// Posada en marxa del servidor
app.listen(8080);
```

> **IMPORTANT**: és important l'ordre en què apliquem el *middleware*, com hem dit abans. En primer lloc, carreguem `express.json()`, perquè s'aplique a TOTES les rutes que ho necessiten, després carreguem Bootstrap, i després els encaminadors. Si carreguem primer els encaminadors, per exemple, llavors no tindran disponible ni Bootstrap ni `express.json()`.

2.2. Vista per a llistat general

Anem ara a crear la carpeta `views`, i dins definim una vista anomenada `contactes_llistat.njk`. Dins d'aquesta vista definirem el codi HTML que tindrà, deixant un buit per a mostrar el llistat de contactes:

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="/css/bootstrap.min.css">
    </head>
    <body>
        <div class="container">
            <h1>Llistat de contactes</h1>
            <div>
                <!-- Ací mostrarem el llistat -->
            </div>
        </div>
    </body>
</html>
```

Per a poder mostrar el llistat de contactes, necessitem proporcionar aquest llistat a la vista. Això ho farem des de la ruta de consulta de contactes. En sessions anteriors havíem establit aquesta ruta en l'arxiu `routes/contactes.js`, sota la URI `GET /contactes`. En aqueixa sessió retornava els contactes en format JSON, però ara li direm simplement que mostre (renderitze) la vista del llistat de contactes. Així ens quedarà ara aquest encaminador:

```
router.get('/', (req, res) => {
    res.render('contactes_llistat');
});
```

Notar que, per a mostrar una vista, n'hi ha prou que indiquem el nom de l'arxiu, sense l'extensió. Nunjucks ja s'encarrega de localitzar l'arxiu, processar el contingut dinàmic que tinga i enviar el resultat.

No obstant això, ens falta alguna cosa en l'encaminador anterior. Necessitem poder facilitar-li a la vista el llistat de contactes. Per a això, utilitzarem Mongoose per a obtindre aquest llistat, i una vegada obtingut, renderitzarem la vista amb `render`, passant com segon paràmetre les dades que necessita la vista per a treballar (el llistat de contactes, en aquest cas).

```
router.get('/', (req, res) => {
    Contacte.find().then(resultat => {
        res.render('contactes_llistat', {contactes: resultat});
    }).catch(error => {
        // Ací podríem renderitzar una pàgina d'error
    });
});
```

Finalment, en la plantilla, podem reemplaçar el comentari que hem deixat de "Ací mostrarem el llistat" amb el llistat efectiu, amb aquest codi:

```
<ul>
    {% for contacte in contactes %}
        <li>{{ contacte.nom }}</li>
    {% endfor %}
</ul>
```

Hem emprat la clàusula `for` de Nunjucks per a iterar sobre una col·lecció d'elements (la col·lecció `contactes` que rebem de l'encaminador). Per a cada element, vam mostrar un item de llista, i el nom de cada contacte en ell. Observeu la notació de la doble clau `{{ ... }}` per a mostrar informació dels objectes amb els quals estem treballant (en aquest cas, cada contacte de la llista que rebem de l'encaminador).

2.3. Algunes qüestions addicionals

La clàusula `for` que hem emprat abans disposa d'algunes utilitats més. Dins de l'element `loop`, disposem d'algunes propietats que podem consultar en cada iteració, com per exemple:

- `index`: que obté la posició en la col·lecció de l'element que s'està explorant actualment, començant per 1.
- `index0`: similar a l'anterior, però començant a comptar per 0.
- `first`: una propietat booleana que és una certa quan estem en el primer element de la col·lecció
- `last`: una propietat booleana que és una certa quan estem en l'últim element de la col·lecció
- `length`: que obté el nombre total d'elements de la col·lecció

Es té disponible també una clàusula `if` per a comprovar condicions, i els operadors `and` i `or` per a enllaçar condicions. A més, es té una clásula `elif` per a enllaçar amb `if` i comprovar altres condicions, i també una clàusula `else` que serveix tant per a mostrar un últim camí en seqüències *if..elif..*, com per a mostrar què fer en un `for` si no hi ha elements.

Per exemple, així podríem mostrar el llistat de contactes amb estils diferents per als items pares i imparells, i amb un missatge personalitzat si no hi ha elements que mostrar.

```
<ul>
    {% for contacte in contactes %}
        {% if loop.index % 2 == 0%}
            <li class="parell">
        {% else %}
            <li class="imparell">
        {% endif %}
        {{ contacte.nom }}</li>
    {% else %}
        <li>No hi ha contactes que mostrar.</li>
    {% endfor %}
</ul>
```

3. Definició de vistes jeràrquiques i inclusions

En l'exemple realitzat abans definim una vista `contactes_llistat.njk` utilitzant el motor de plantilles Nunjucks per a mostrar el llistat de contactes de la base de dades. A mesura que l'aplicació creix i necessitem anar definint més i més vistes, podem deduir que la forma en què ho hem fet en l'exemple anterior té alguns desavantatges importants. Per exemple, i sobretot, el seu modularidad. Si, per exemple, tenim 20 vistes definides com la del llistat de contactes anterior, i decidim canviar el menú d'enllaços, hauríem d'editar-lo en les 20 vistes. El mateix ocorreria si volem canviar la informació de l'encapçalat o el peu, entre altres coses, que sol ser comuna a totes les pàgines.

Per a evitar aquest inconvenient Nunjucks (i molts altres motors de plantilles) permeten realitzar un **disseny jeràrquic** d'aquestes. És a dir, podem crear una o diverses plantilles base amb el contingut general que tindran un conjunt de vistes, i fer que aquestes vistes "hereten" d'aquestes plantilles per a definir únicament aquell contingut que els és propi, incloent automàticament el contingut heretat.

3.1. Herència de plantilles

Farem un exemple amb l'aplicació de contactes "*ContactesWeb*" que venim desenvolupant en aquesta sessió. Creem en la carpeta `views` una plantilla anomenada `base.njk`, que tindrà el contingut general de qualsevol vista de l'aplicació: l'encapçalat (*head*) amb els estils i arxius JavaScript per a la part client, el menú de l'aplicació, i el peu de pàgina, si n'hi ha. Cada vista canviarà el títol de pàgina, i el contingut principal d'aquesta vista. Per a deixar aquestes dues seccions obertes i que es puguen modificar en cada vista es defineixen *blocs*, associant a cada bloc un nom. Així, la nostra plantilla `base.njk` pot quedar així:

```
<!DOCTYPE html>
<html>
    <head>
        <title>{% block titol %} {% endblock %}</title>
        <link rel="stylesheet" href="/css/bootstrap.min.css">
    </head>
    <body>
    <div class="container">
        {% block contingut %}
        {% endblock %}
    </div>
    </body>
</html>
```

Com podem veure, amb l'estructura `block` definim blocs de continguts, associats a un nom, de manera que tot el que queda fora d'aqueixos blocs és fix per a totes les vistes que hereten de la plantilla.

Ara la nostra pàgina de `contactes_llistat.njk` només ha de limitar-se a heretar d'aquesta plantilla, i definir el contingut dels dos blocs. Pot quedar així:

```
{% extends "base.njk" %}

{% block titol %}Contactes | Llistat{% endblock %}

{% block contingut %}

 <h1>Llistat de contactes</h1>

 <ul>
    {% for contacte in contactes %}
        <li>{{ contacte.nom }}</li>
    {% endfor %}
 </ul>

{% endblock %}
```

De la mateixa manera, podríem definir altres vistes, com per exemple la fitxa d'un contacte (`contacte_fitxa.njk`):

```
{% extends "base.njk" %}

{% block titol %}Contactes | Fitxa{% endblock %}

{% block contingut %}

 <h1>Fitxa d'un contacte</h1>

 <p><strong>Nom:</strong> {{ contacte.nom }}</p>
 <p><strong>Edat:</strong> {{ contacte.edat }}</p>
 <p><strong>Telèfon:</strong> {{ contacte.telefon }}</p>

{% endblock %}
```

> **NOTA**: en el cas de la fitxa del contacte, caldria modificar el contingut de l'encaminador (mètode *GET /contactes/:id*) perquè renderitze la vista i li passe l'objecte `contacte` trobat.

3.2. Inclusió de plantilles

Una altra funcionalitat realment útil que proporcionen molts motors de plantilles és la possibilitat d'incloure el contingut d'una plantilla directament en una altra, com si poguérem fer un "còpia-pega" directament de l'una en l'altra. Això evita haver de duplicar el codi HTML en les plantilles i, novament, facilitar la possibilitat de posteriors canvis.

Per a incloure una vista o plantilla dins d'una altra, emprarem la instrucció `include`. Per exemple, si volem incloure una vista amb el menú de navegació de la web, podem fer això, just en el lloc on volem situar el menú:

```
{% include "menu.njk" %}
```

3.3. Pàgines d'error

En alguns casos ens pot interessar mostrar alguna pàgina d'error. Per exemple, quan les dades d'un contacte no s'hagen trobat, o quan no haja sigut possible realitzar alguna operació (un esborrat, o una inserció, per exemple). Alguns frameworks automatitzen aquests processos permetent crear pàgines amb un codi d'error determinat però, en el cas d'Express, aquesta gestió es deixa més oberta.

Podem, per exemple, crear una vista `error.njk` en la nostra carpeta `views` amb una estructura general per a mostrar errors en l'aplicació:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Error</title>
        <link rel="stylesheet" href="/css/bootstrap.min.css">
    </head>
    <body>
        <div class="container">
            <h1>Error</h1>
            <div class="alert alert-danger">
                {% if error %}
                    {{ error }}
                {% else %}
                    Error en l'aplicació
                {% endif %}
            </div>
        </div>
    </body>
</html>
```

Ens quedaria renderitzar aquesta vista des dels apartats on es detecte un error. Per exemple, en el llistat de contactes o la fitxa d'un contacte:

```
// Servei de llistat general
router.get('/', (req, res) => {
    Contacte.find().then(resultat => {
        res.render('contactes_llistat', {contactes: resultat});
    }).catch(error => {
        res.render('error', {error: 'Error llistant contactes'});
    });
});

// Servei de llistat per id
router.get('/:id', (req, res) => {
    Contacte.findById(req.params['id']).then(resultat => {
        if(resultat)
            res.render('contactes_fitxa', {contacte: resultat});
        else
            res.render('error', {error: 'Contacte no trobat'});
    }).catch(error => {
        res.render('error', {error: 'Error buscant contacte'});
    });
});
```

En l'exemple d'aquest document pots consultar el projecte *ContactesWeb* amb aquests elements ja implementats per a poder-los provar.

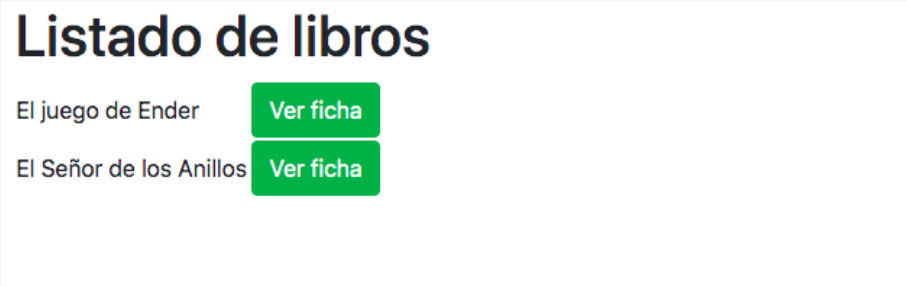
Exercici 1:

Crea un projecte **LlibresWeb** que siga una còpia de l'exercici *LlibresREST_v2* de sessions anteriors. Instal·la Nunjucks i Bootstrap en el nou projecte, i deixa un arxiu principal `index.js` similar al de l'exemple de contactes, carregant les llibreries, els encaminadors que hi haja, etc, i configurant Nunjucks com el motor de plantilles de l'aplicació.

Crea una plantilla anomenada `base.njk` que incorpore els elements comuns a les vistes que realitzarem: un encapçalat *head* amb la inclusió d'estils Bootstrap, peu de pàgina, i un menú de navegació. Deixa dos blocs (*block*) editables anomenats "titol" i "contingut", com en l'exemple de contactes.

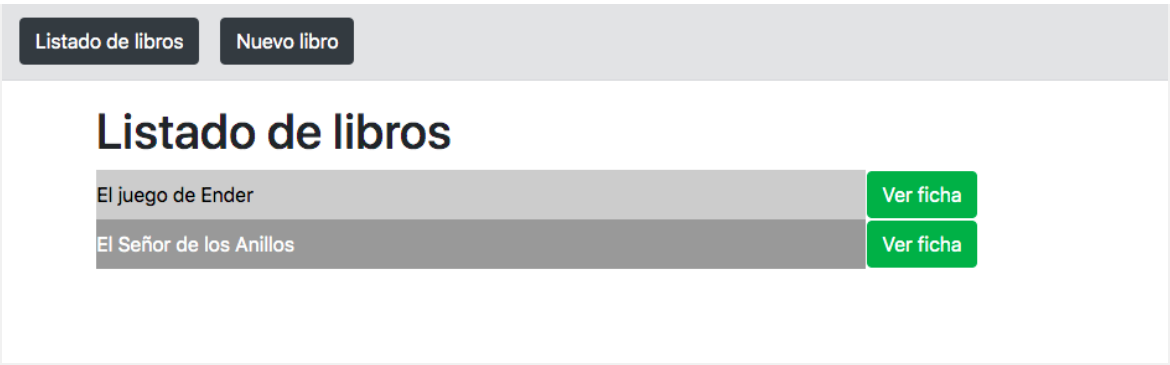
Definirem dues vistes, associades als dos encaminadors de llistat de llibres i fitxa de llibres, que modificarem. Recorda que hauràs de crear els arxius de les vistes en la carpeta `views` del projecte:

- `llibres_llistat.njk` : heretarà de *base.njk* i rebrà de l'encaminador GET per a `/llibres` el llistat complet de llibres, i mostrarà el llistat de llibres per pantalla. S'haurà de mostrar el títol del llibre, i al seu costat, un enllaç per a anar a la fitxa detallada del llibre. Aquesta vista pot quedar-te més o menys així. Pots usar `class="btn btn-success"` en l'enllaç per a mostrar-lo amb aqueixa aparença de botó en verd, gràcies a Bootstrap. També pots, opcionalment, fer que cada fila (parell i imparell) es mostre amb colors alterns.



- `llibres_fitxa.njk` : també heretarà de *base.njk* i rebrà de l'encaminador GET per a `/llibres/:id` les dades del llibre a mostrar. Es mostraran les seues dades amb un format com el que s'indica a continuació

- `menu.njk` : mostrarà un menú de navegació amb dos enllaços: un per a anar al llistat de llibres (URL `/llibres`), i un altre per a anar al formulari d'inserció de llibres que implementarem més endavant (URL `/llibres/nou`). Inclou amb `include` aquesta vista en la plantilla base, just abans del bloc de "contingut". Aquesta és l'aparença que pot tindre més o menys la vista de llistat ara:



- A més de les vistes anteriors, crea una nova vista anomenada `error.njk` en la carpeta de vistes, que rebrà com a paràmetre un missatge d'error i el mostrarà per pantalla, com l'exemple que s'ha fet abans per als contactes, mostrant un missatge genèric "Error en l'aplicació" si no rep un missatge d'error concret. Modifica els encaminadors GET de llibres perquè, en cas d'error, es renderitze aquesta vista amb el missatge adequat.

NOTA: recorda que els antics encaminadors que retornaven dades JSON hauran de modificar-se per a renderitzar les vistes corresponents en aquest exercici. La resta d'encaminadors que encara no hem tocat (inserció o esborrat de llibres, per exemple), deixa'ls com estaven de la sessió anterior, i ja anirem modificant-los després.