Introducció als serveis REST


Els serveis REST s'han convertit en una arquitectura d'aplicacions molt popular a l'hora d'intercanviar informació entre el costat client i servidor, per la seua versatilitat, ja que permeten que diferents tipus de clients (aplicacions mòbils, d'escriptori o web) puguen sol·licitar informació a un mateix servidor, i utilitzar-la de forma adaptada a cada aplicació. Veurem en aquest document les bases sobre les quals se sustenta.

1. El protocol HTTP i les URL

Per al que veurem a partir d'aquesta sessió, convé tindre clars alguns conceptes de base. Per a començar, qualsevol aplicació web es basa en una arquitectura client-servidor, on un servidor queda a l'espera de connexions de clients, i els clients es connecten als servidors per a sol·licitar uns certs recursos.

Aquestes comunicacions es realitzen mitjançant un protocol anomenat **HTTP** (o HTTPS, en el cas de comunicacions segures). En tots dos casos, client i servidor s'envien una certa informació estàndard, en cada missatge:

- Quant als **clients**, envien al servidor les dades del recurs que sol·liciten, juntament amb una certa informació addicional, com per exemple les capçaleres de petició (informació relativa al tipus de client o navegador, contingut que accepta, etc), i paràmetres addicionals anomenats normalment *dades del formulari*.
- Pel que respecta als **servidors**, accepten aquestes peticions, les processen i envien de tornada algunes dades rellevants, com un codi d'estat (indicant si la petició va poder ser atesa satisfactòriament o no), capçaleres de resposta (indicant el tipus de contingut enviat, grandària, idioma, etc), i el recurs sol·licitat pròpiament dit, si tot ha anat correctament.

Per a sol·licitar els recursos, els clients es connecten o sol·liciten una determinada **URL** (sigles en anglés de "localització uniforme de recursos", *Uniform Resource Location*). Aquesta URL consisteix en una línia de text amb tres seccions diferenciades:

- El **protocol** emprat (HTTP o HTTPS)
- El **nom de domini**, que identifica al servidor i el localitza en la xarxa.
- La **ruta** cap al recurs sol·licitat, dins del propi servidor. Aquesta última part també sol denominar-se **URI** (identificador uniforme de recurs, o en anglés, *Uniform Resource Identifier*). Aquesta URI identifica unívocament el recurs buscat entre tots els altres recursos que puga albergar el servidor.

Per exemple, la següent podria ser una URL vàlida:

http://servidor.com/llibres?id=123

El protocol emprat és *http*, i el nom de domini és *servidor.com*. Finalment, la ruta o URI és *llibres?id=123*, i el text després de l'interrogant '?' és la informació addicional anomenada *query string* o *dades del formulari*.

Aquesta informació permet aportar una mica més d'informació sobre el recurs sol·licitat. En aquest cas, podria fer referència al codi del llibre que estem buscant. Depenent de com s'haja implementat el servidor, també podríem reescriure aquesta URL d'aquesta altra manera, amb el mateix significat:

http://servidor.com/llibres/123

2. Els serveis REST

En aquesta secció del curs veurem com aplicar l'aprés fins ara per a desenvolupar un servidor senzill que proporcione una API REST als clients que es connecten. **REST** són les sigles de *REpresentational State Transfer*, i designa un estil d'arquitectura d'aplicacions distribuïdes, com les aplicacions web. En un sistema REST, identifiquem cada recurs a sol·licitar amb una URI (identificador uniforme de recurs), i definim un conjunt delimitat de comandos o mètodes a realitzar, que típicament són:

- **GET**: per a obtindre resultats d'algun tipus (llistats complets o filtrats per alguna condició)
- **POST**: per a realitzar insercions o afegir elements en un conjunt de dades
- **PUT**: per a realitzar modificacions o actualitzacions del conjunt de dades
- **DELETE**: per a realitzar esborrats del conjunt de dades

Existeixen altres tipus de comandos o mètodes, com per exemple PATCH (similar a PUT, però per a canvis parcials), HEAD (per a consultar només l'encapçalat de la resposta obtinguda), etc. Ens centrarem, no obstant això, en els quatre mètodes principals anteriors

Per tant, identificant el recurs a sol·licitar (URI) i el comando a aplicar-li, el servidor que ofereix aquesta API REST proporciona una resposta a aqueixa petició. Aquesta resposta típicament ve donada per un missatge en format JSON o XML (encara que aquest últim cada vegada està més en desús).

Veurem com podem identificar els diferents tipus de comandos de nostra API, i les URIs dels recursos a sol·licitar, per a després donar una resposta en format JSON davant cada petició.

2.1. Els *endpoints*

Un *endpoint*, aplicat a serveis REST, consisteix en la URI i la comanda que s'ha de sol·licitar al servidor per a un determinat servei. Per exemple, si hem d'accedir a la URI `/llibres` per GET per a obtindre un llistat de llibres, l'*endpoint* corresponent seria:

```
GET /llibres
```

És convenient recordar aquesta nomenclatura, perquè és bastant habitual trobar-la quan treballem amb serveis REST.

3. El format JSON

JSON són les sigles de *JavaScript Object Notation*, una sintaxi pròpia de JavaScript per a poder representar objectes com a cadenes de text, i poder així serialitzar i enviar informació d'objectes a través de fluxos de dades (arxius de text, comunicacions client-servidor, etc).

Un objecte JavaScript es defineix mitjançant una sèrie de propietats i valors. Per exemple, les dades d'una persona (com a nom i edat) podríem emmagatzemar-los així:

```
let persona = {
    nom: "Nacho",
    edat: 39
};
```

Aquest mateix objecte, convertit a JSON, formaria una cadena de text amb aquest contingut:

```
{"nom":"Nacho","edat":39}
```

De la mateixa manera, si tenim una col·lecció (vector) d'objectes com aquesta:

```
let persones = [
    { nom: "Nacho", edat: 39},
    { nom: "Mario", edat: 4},
    { nom: "Laura", edat: 2},
    { nom: "Nora", edat: 10}
];
```

Transformada a JSON segueix la mateixa sintaxi, però entre claudàtors:

```
[{"nom":"Nacho","edat":39}, {"nom":"Mario","edat":4},
 {"nom":"Laura","edat":2}, {"nom":"Nora","edat":10}]
```

JavaScript ofereix un parell de mètodes útils per a convertir dades a format JSON i viceversa. Aquests mètodes són `JSON.stringify` (per a convertir un objecte o array JavaScript a JSON) i `JSON.parse` (per al procés invers, és a dir, convertir una cadena JSON en un objecte JavaScript). Ací veiem un exemple de cadascun:

```
let persones = [
    { nom: "Nacho", edat: 39},
    { nom: "Mario", edat: 4},
    { nom: "Laura", edat: 2},
    { nom: "Nora", edat: 10}
];

// Convertir array a JSON
let personesJSON = JSON.stringify(persones);
console.log(personesJSON);

// Convertir JSON a array
let persones2 = JSON.parse(personesJSON);
console.log(persones2);
```

En els següents exemples realitzarem comunicacions client-servidor on el client sol·licitarà al servidor una sèrie de serveis, i aquest respondrà retornant un contingut en format JSON. No obstant això, gràcies al framework Express que utilitzarem, la conversió des d'un format a un altre serà automàtica, i no haurem de preocupar-nos d'utilitzar aquests mètodes de conversió.

3.1. JSON i serveis REST

Com comentàvem abans, JSON és hui dia el format més utilitzat per a donar resposta a peticions de serveis REST. El seu altre "competidor", el format XML, està cada vegada més en desús per a aquestes tasques.

A l'hora d'emetre una resposta a un servei utilitzant format JSON, és habitual que aquesta tinga un format determinat. En general, en les respostes que emetem a partir d'ara per a serveis REST en el curs, utilitzarem una estructura general basada en:

- Una dada booleana (podem cridar-lo `ok`, per exemple), que indique si la petició s'ha pogut atendre satisfactòriament o no.
- Un missatge d'error (podem anomenar-ho `error`, per exemple), que serà present únicament si l'anterior dada booleana és fals, la qual cosa indicaria que la petició no s'ha pogut resoldre.
- Les dades de resposta, que seran presents només si la dada booleana és vertader, la qual cosa indica que la petició s'ha pogut atendre satisfactòriament. Notar que aquestes dades de resposta poden ser un text, un objecte simple JavaScript, o un array d'objectes.

Addicionalment, com veurem en els exemples a continuació, també és recomanable afegir a la resposta un codi d'estat HTTP, que indique si s'ha pogut servir satisfactòriament o hi ha hagut algun error.