

Tema 7

APIs REST con

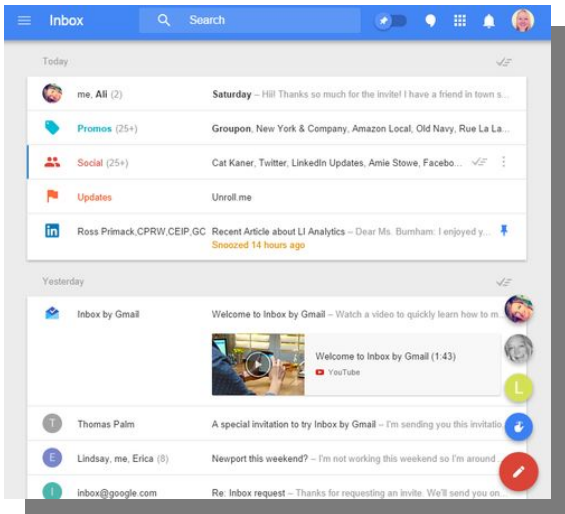
Express

Micael Gallego
micael.gallego@gmail.com
@micael_gallego

- **Introducción**
- Formato JSON
- Funcionamiento de un servicio REST
- Clientes de servicios REST
- APIs REST con Express

- En una aplicación web, el cliente (**navegador**) se comunica con el servidor (**servidor web**) usando el protocolo **http**
- En una aplicación web sin AJAX, las peticiones **http** devuelven un **documento HTML** que será **visualizado** por el navegador
- En las aplicaciones con AJAX y las aplicaciones SPA, las peticiones **http** se utilizan para intercambiar **información** entre el navegador y el servidor (pero no HTML)

- Ejemplo de aplicación SPA haciendo peticiones **http** a un servidor para obtener **información**

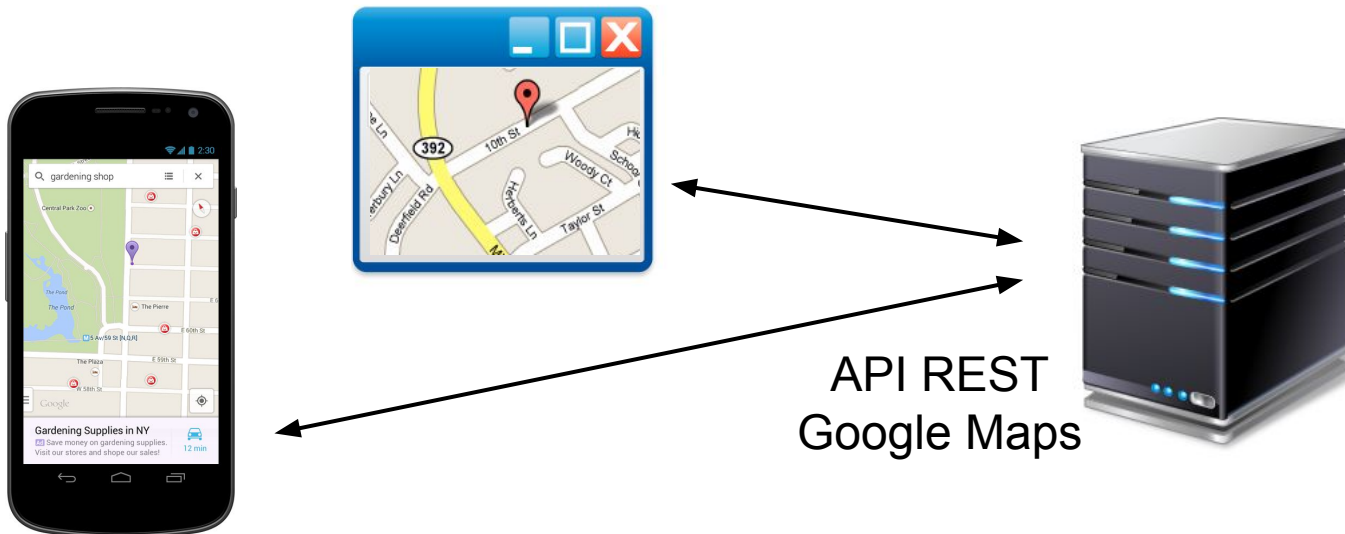


`http://www.miweb.com/users/34`

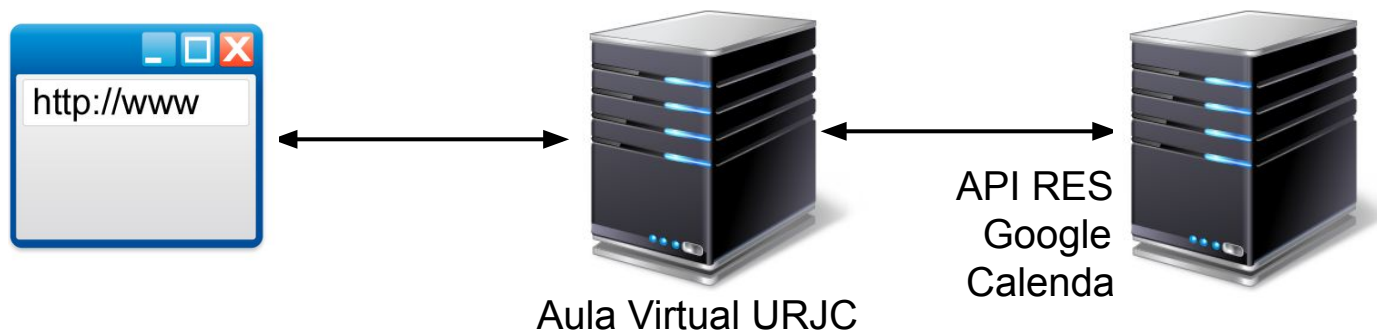
```
{  
  "name": "Pepe",  
  "surname": "López",  
  "age": 45,  
  "email": "pepe@miweb.com",  
  "friends": [ "María", "Juan" ]  
}
```



- Además de un **navegador web**, otros tipos de aplicaciones también usan las **APIs REST**
 - **Otros clientes:** Apps móviles, TVs, consolas...
 - Ejemplo: La aplicación de **Google Maps** para Android es un cliente de la misma **API REST** de la web SPA

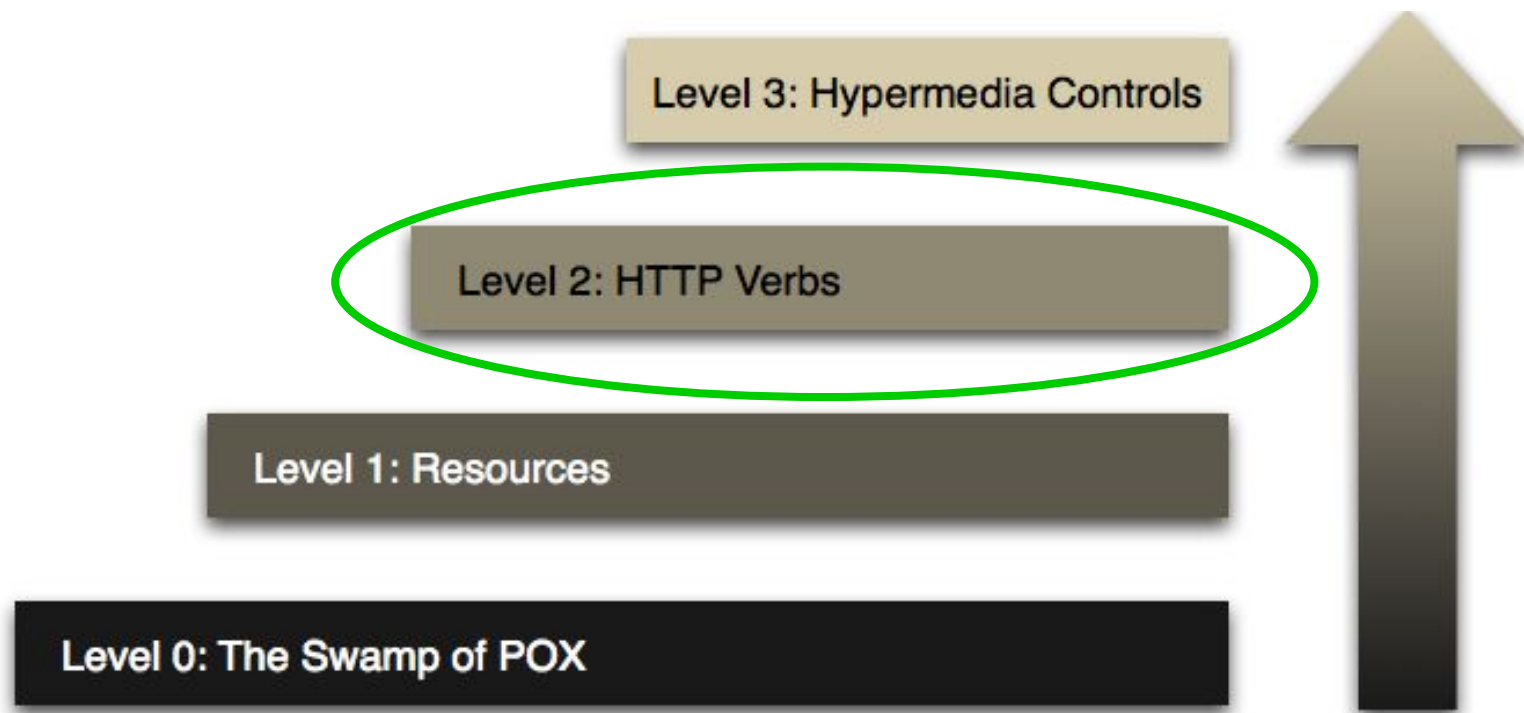


- Además de un **navegador web**, otros tipos de aplicaciones también usan las **APIs REST**
 - **Otros servidores:** El backend de una aplicación web puede usar APIs REST además de sus bases de datos para ofrecer sus servicios
 - Ejemplo: El Aula Virtual de la URJC podría usar la API REST de Google Calendar para publicar eventos



- REST es acrónimo de ***REpresentational State Transfer***, Transferencia de Estado Representacional.
- El término se acuñó en el año 2000, en la tesis doctoral de **Roy Fielding**, uno de los principales autores de la especificación del protocolo **HTTP**
- Existen otros tipos de **servicios web** como **SOAP** basados en **XML** y mucho **más complejos**, pero no se usan tanto como los servicios web REST

- Niveles de cumplimiento de los principios REST



<http://martinfowler.com/articles/richardsonMaturityModel.html>

- Un **servicio REST** ofrece operaciones **CRUD** (**creación, lectura, actualización y borrado**) sobre recursos (items de información) del servidor web
- Se aprovecha de todos los aspectos del **protocolo http**: URL, métodos, códigos de estado, cabeceras...
- La información se intercambia en formato **JSON** (o XML)

- Introducción
- **Formato JSON**
- Funcionamiento de un servicio REST
- Clientes de servicios REST
- APIs REST con Express

- Acrónimo de *JavaScript Object Notation*
- Es un subconjunto de la notación literal de objetos de **JavaScript**
- Se procesa de forma muy rápida en **JavaScript**



<http://www.json.org/>

Información estructurada con JSON

```
{ "menu": {  
  "id": 1,  
  "value": "File",  
  "enabled": true,  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick": "CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}}
```

- **JSON** se utiliza para la codificación de la información en la mayoría de los servicios REST(aunque también se puede usar **XML**).
- También se usa para **estructurar** cualquier tipo de información:
 - **Ficheros de configuración**
 - Datos en disco
 - Bases de datos NoSQL (Mongo)

- Introducción
- Formato JSON
- **Funcionamiento de un servicio REST**
- Clientes de servicios REST
- APIs REST con Express

- El enfoque más habitual en los servicios REST es el nivel 2:
 - Los recursos se identifican en la **URI**
<http://server/anuncios/vendo-moto-23-10-2014>
 - Las **operaciones** que se quieren realizar con ese recurso son los **métodos del protocolo HTTP**
 - La información se devuelve codificada en **JSON**
 - Se usan los códigos de **estado http** para notificar errores (p.e. 404 Not found)

- Los recursos se identifican en la URI
 - Parte de la URL es fija y otra parte apunta al recurso concreto
 - Ejemplos:
 - <http://server/anuncios/vendo-moto-23-10-2014>
 - <http://server/users/bob>
 - <http://server/users/bob/anuncios/comparto-piso>
 - <http://server/users/bob/anuncios/44>

<https://restfulapi.net/rest-api-design-tutorial-with-example/>

- **Las operaciones se codifican como métodos http**
 - **GET:** Devuelve el recurso, generalmente codificado en JSON. No envían información en el cuerpo de la petición.
 - **DELETE:** Borra el recurso. No envían información en el cuerpo de la petición.
 - **POST:** Añade un nuevo recurso. Envía el recurso en el cuerpo de la petición.
 - **PUT:** Modifica el recurso. Habitualmente se envía el recurso obtenido con GET pero modificando los campos que se consideren (existen optimizaciones)

- La información se devuelve codificada en JSON
 - Petición:
 - **URL:** <http://server/bob/bookmarks/6>
 - **Método:** GET
 - Respuesta:
 - **Header:** content-type: application/json
 - **Body:**

```
{  
  id: 6,  
  uri: "http://bookmark.com/2/bob",  
  description: "A description"  
}
```

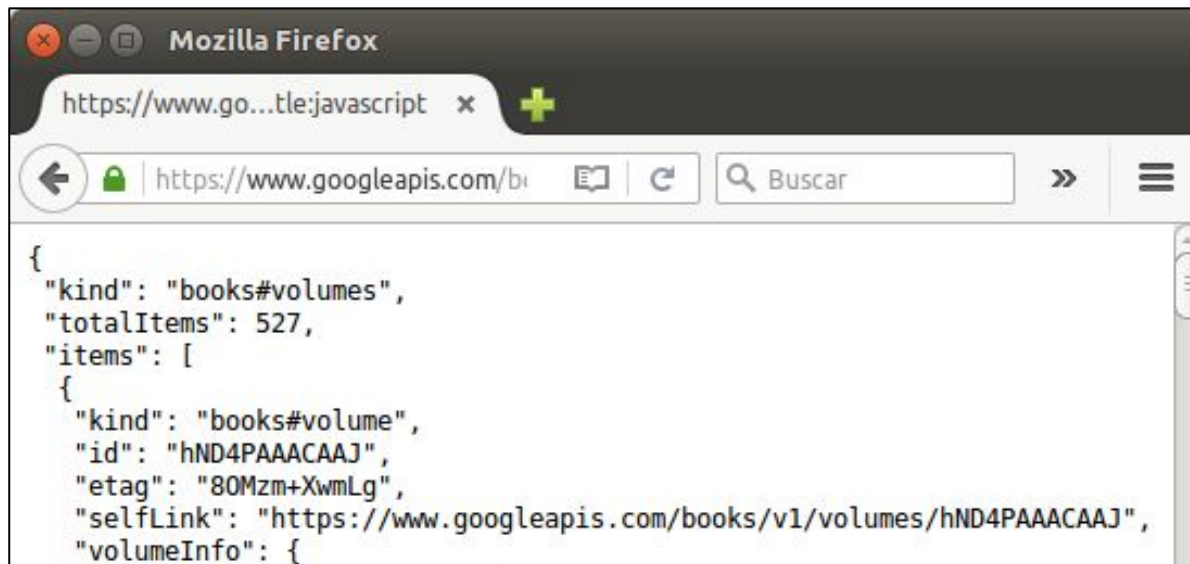
- Se usan los códigos de estado http para notificar errores:
 - **100-199:** No están definidos. Describen fases de ejecución de la petición.
 - **200-299:** La petición fue procesada correctamente.
 - **300-399:** El cliente debe hacer acciones adicionales para completar la petición, por ejemplo, una redirección a otra página.
 - **400-499:** Se usa en casos en los que el cliente ha realizado la petición incorrectamente (404 No existe).
 - **500-599:** Se usa cuando se produce un error procesando la petición.

- Introducción
- Formato JSON
- Funcionamiento de un servicio REST
- **Cientes de servicios REST**
- APIs REST con Express

- Los servicios REST están diseñados para ser utilizados por **aplicaciones** (no por humanos)
- Todos los **lenguajes de programación** disponen de librerías para uso de servicios REST (**JavaScript**, Java...)
- Como desarrolladores podemos usar **herramientas interactivas** para hacer pruebas (hacer peticiones y ver las respuestas)

- Herramientas interactivas

- El navegador web es una herramienta básica que se puede usar para hacer peticiones GET a APIs REST



<https://www.googleapis.com/books/v1/volumes?q=intitle:javascript>

- **Herramientas interactivas**

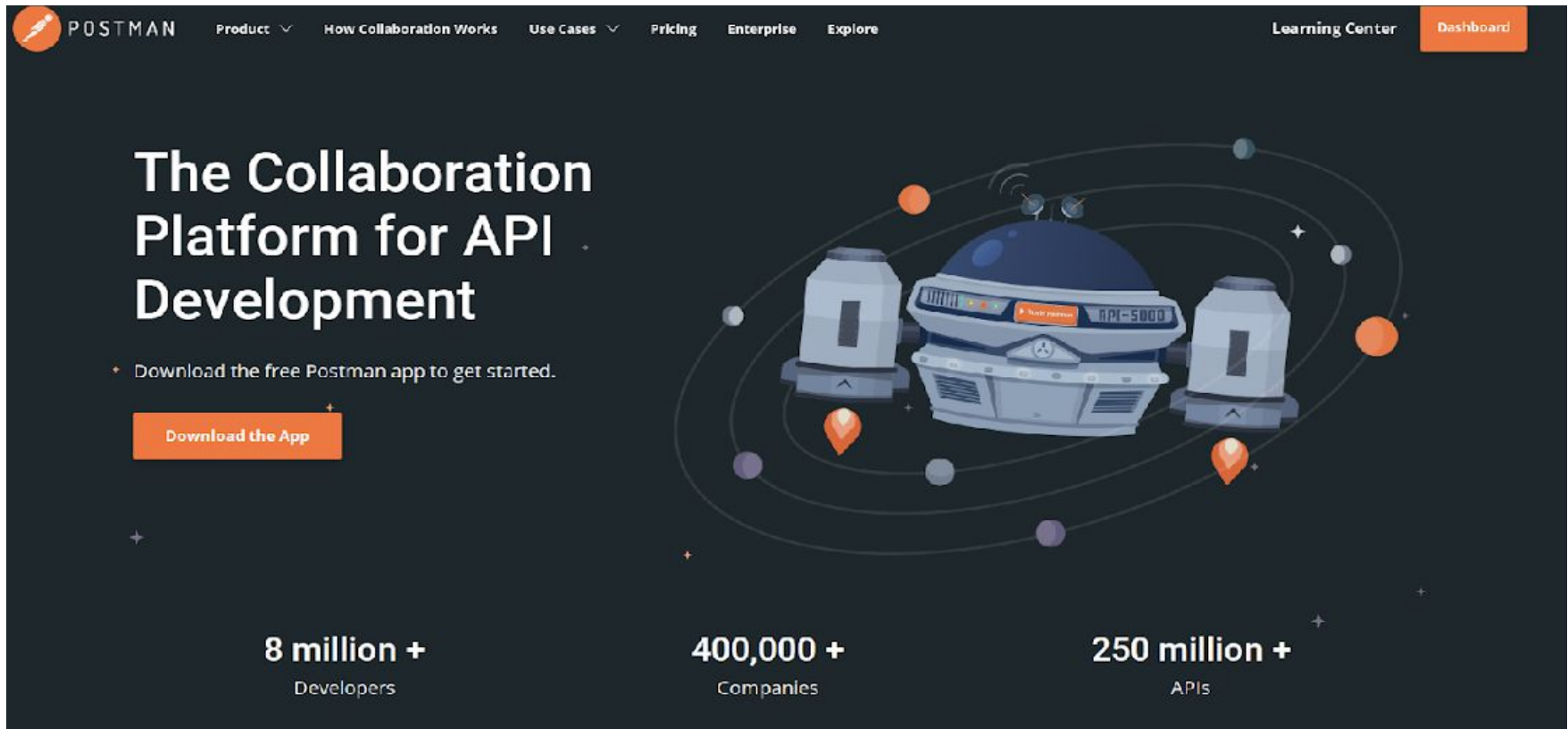
- Tipos

- Integradas en el entorno de desarrollo
- Extensiones del navegador

- Permiten

- Realizar peticiones REST con cualquier método (GET, POST, PUT...)
- Especificar URL, cabeceras (headers)...
- Analizar la respuesta: Cuerpo, status http...

- Postman - REST Client

The image shows the Postman website landing page. At the top, there is a navigation bar with the Postman logo and links for Product, How Collaboration Works, Use Cases, Pricing, Enterprise, and Explore. On the right side of the navigation bar are links for Learning Center and a Dashboard button. The main content area features the headline "The Collaboration Platform for API Development" and a sub-headline "Download the free Postman app to get started." with a "Download the App" button. To the right of the text is an illustration of a futuristic space station or satellite with two smaller modules orbiting it. At the bottom of the page, three statistics are displayed: "8 million + Developers", "400,000 + Companies", and "250 million + APIs".

POSTMAN Product ▾ How Collaboration Works Use Cases ▾ Pricing Enterprise Explore Learning Center Dashboard

The Collaboration Platform for API Development

• Download the free Postman app to get started.

Download the App

8 million +
Developers

400,000 +
Companies

250 million +
APIs

<https://www.getpostman.com/>

Postman

Search

History Collections

Today

- GET <https://www.googleapis.com/books/v1/volumes?q=intitle:javascript>
- GET [http://en.wikipedia.org/w/api.php?format=json&action=query&titles=Main%](http://en.wikipedia.org/w/api.php?format=json&action=query&titles=Main%20Page)

Builder Runner Import

https://www.google... No environment

GET <https://www.googleapis.com/books/v1/volumes?q=intitle:javascript> Params Send

Authorization Headers (0) Body Pre-request script Tests

No Auth

Body Cookies Headers (15) Tests (0/0) Status 200 OK Time 550 ms

Pretty Raw Preview JSON

```
1 {
2   "kind": "books#volumes",
3   "totalItems": 527,
4   "items": [
5     {
6       "kind": "books#volume",
7       "id": "-4zGCQAAQBAJ",
8       "etag": "pnWVrlo/7f0",
9       "selfLink": "https://www.googleapis.com/books/v1/volumes/-4zGCQAAQBAJ",
10      "volumeInfo": {
11        "title": "JavaScript Una Guía de Aprendizaje para el Lenguaje de Progr
12        "authors": [
13          "Troy Dimes"
14        ],
15        "publisher": "Babelcube Inc.",
16        "publishedDate": "2015-06-01",
17        "description": "This book is a comprehensive guide to JavaScript, covering all the
18        "pageCount": 440
19      }
20    }
21  ]
22 }
```

- Vamos a familiarizarnos con las herramientas interactivas de acceso a APIs REST
- Haremos una petición GET a una API REST pública y analizaremos la información obtenida
- Usaremos la API REST de libros de Google Play

<https://www.googleapis.com/books/v1/volumes?q=intitle:javascript>

- Documentación de la API REST
 - <https://developers.google.com/books/docs/v1/using>

• Cliente JavaScript

- Las aplicaciones web con **AJAX** o con arquitectura **SPA**, implementadas con **JavaScript**, usan servicios **REST** desde el navegador
- Se pueden usar APIs REST usando la **API estándar** del browser o **librerías externas** (jQuery)

- **Cliente JavaScript: jQuery**

- Muestra en la consola el resultado de la API REST

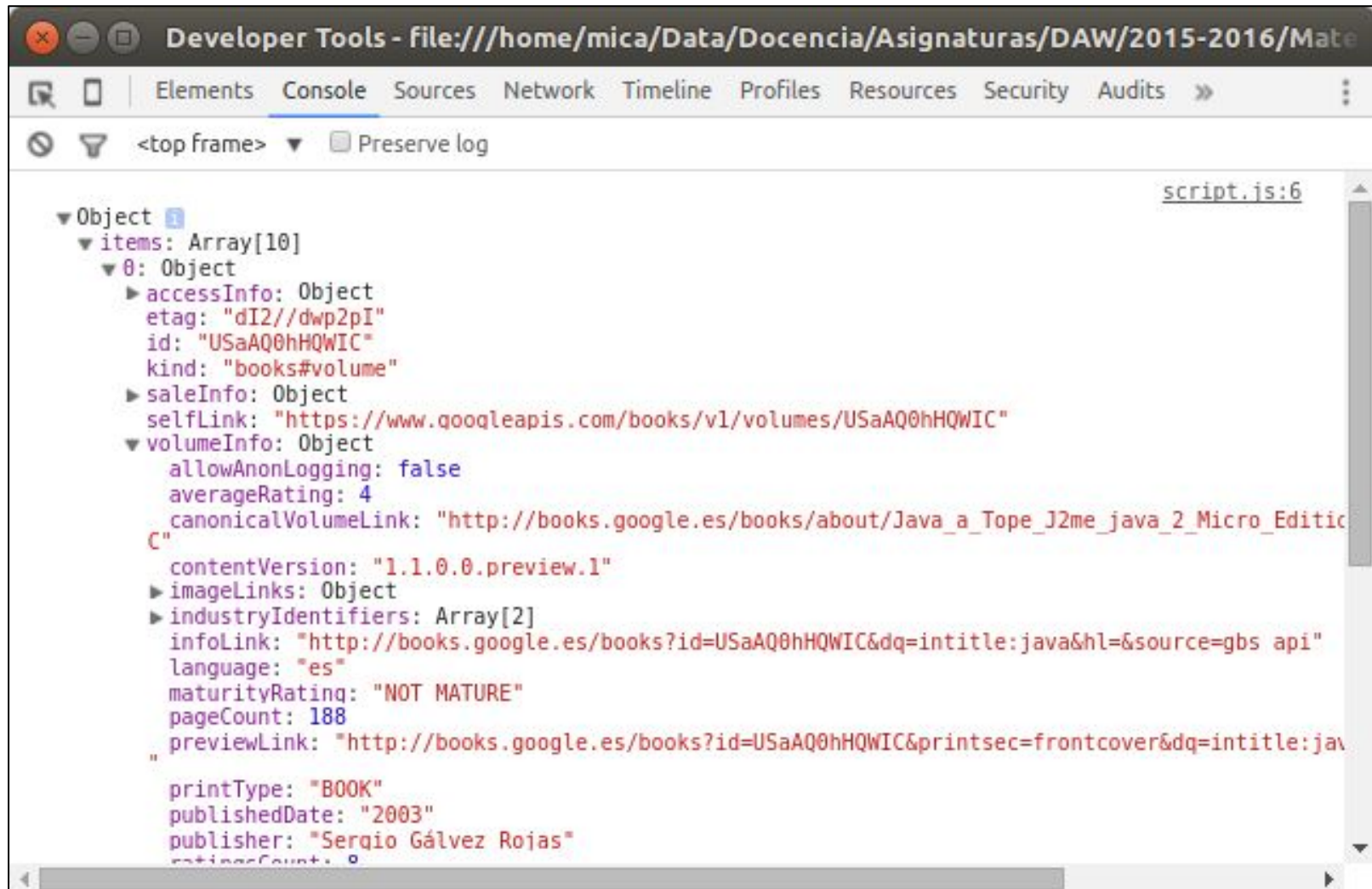
script.js

```
$(document).ready(function(){  
  
    $.ajax({  
        url:"https://www.googleapis.com/books/v1/volumes?q=intitle:java"  
    }).done(function(data) {  
        console.log(data);  
    });  
  
});
```

<http://api.jquery.com/jquery.ajax/>

- Cliente JavaScript: jQuery

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.4.1.min.js">
    </script>
    <script src="script.js"></script>
  </head>
  <body>
  </body>
</html>
```



The screenshot shows the Chrome Developer Tools Console with the 'Console' tab selected. The breadcrumb path is '<top frame>' and the 'Preserve log' checkbox is checked. The log shows a single message from 'script.js:6' which is an expanded JavaScript object. The object has a property 'items' which is an array of 10 objects. The first object in the array is expanded, showing the following properties:

- `accessInfo`: Object
 - `etag`: "dI2//dwp2pI"
 - `id`: "USaAQ0hHQWIC"
 - `kind`: "books#volume"
- `saleInfo`: Object
 - `selfLink`: "https://www.googleapis.com/books/v1/volumes/USaAQ0hHQWIC"
- `volumeInfo`: Object
 - `allowAnonLogging`: false
 - `averageRating`: 4
 - `canonicalVolumeLink`: "http://books.google.es/books/about/Java_a_Tope_J2me_java_2_Micro_Editic"
 - `contentVersion`: "1.1.0.0.preview.1"
 - `imageLinks`: Object
 - `industryIdentifiers`: Array[2]
 - `infoLink`: "http://books.google.es/books?id=USaAQ0hHQWIC&dq=intitle:java&hl=&source=gb&api"
 - `language`: "es"
 - `maturityRating`: "NOT MATURE"
 - `pageCount`: 188
 - `previewLink`: "http://books.google.es/books?id=USaAQ0hHQWIC&printsec=frontcover&dq=intitle:jav"
 - `printType`: "BOOK"
 - `publishedDate`: "2003"
 - `publisher`: "Sergio Gálvez Rojas"
 - `ratingCount`: 0

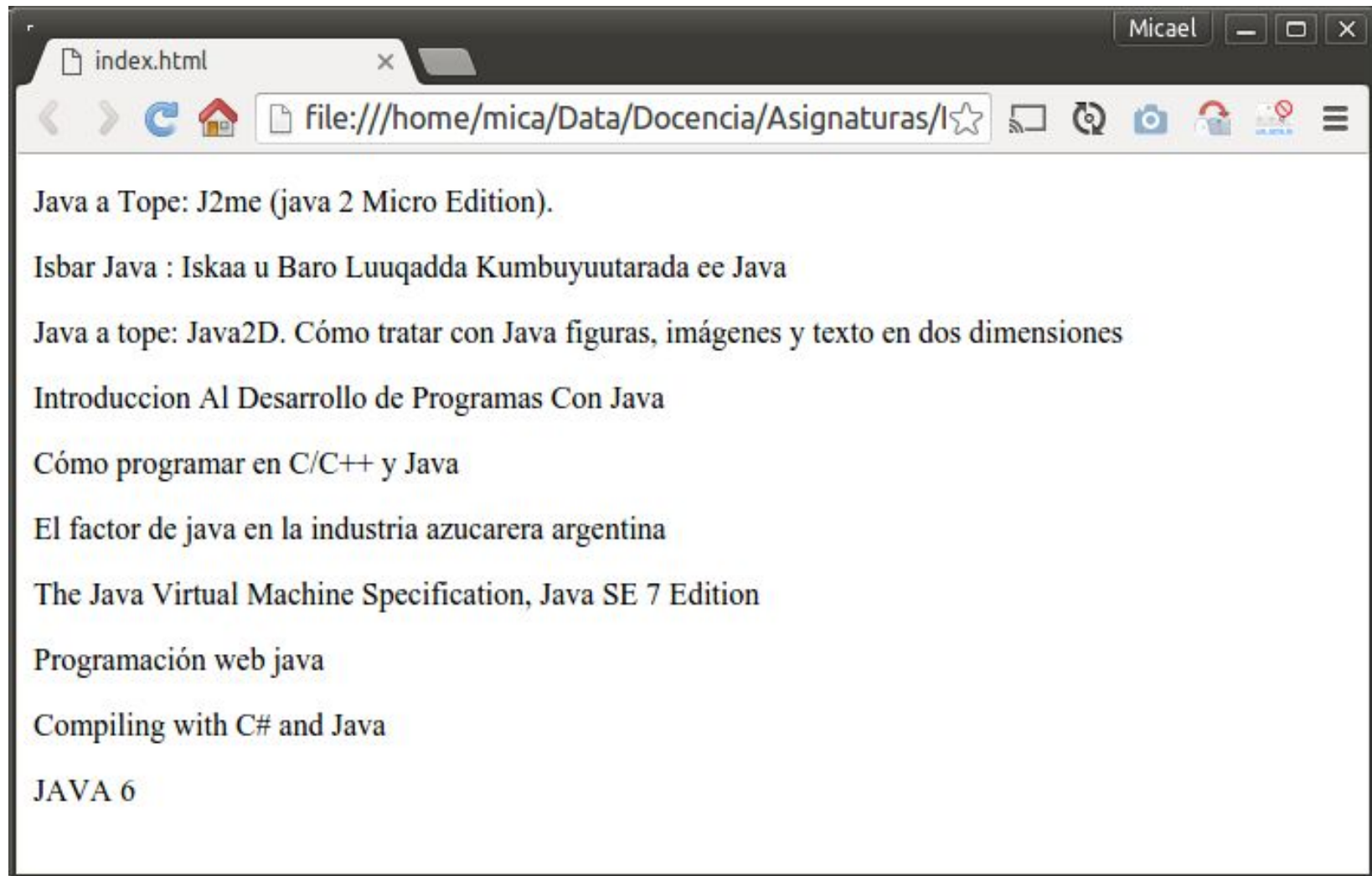
- **Cliente JavaScript: jQuery**

- Muestra en la página los títulos de los libros

script.js

```
$(document).ready(function(){
    $.ajax({
        url:"https://www.googleapis.com/books/v1/volumes?q=intitle:java"
    }).done(function(data) {
        for(var i=0; i<data.items.length; i++){
            $("body").append(
                "<p>" + data.items[i].volumeInfo.title + "</p>");
        }
    });
});
```

<http://api.jquery.com/jquery.ajax/>



- Introducción
- Formato JSON
- Funcionamiento de un servicio REST
- Clientes de servicios REST
- **APIs REST con Express**

- Las librerías básicas de Node permiten crear una API REST
- Pero son muy limitadas y se suele usar el paquete NPM **express**

express

<https://expressjs.com>

Crear una API REST básica

```
var express = require('express');

var app = express();

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => {
  console.log('Example app listening on port 3000!');
});
```

<http://localhost:3000>

APIs REST con Express

- Implementar diferentes operaciones de la API REST

```
app.METHOD(PATH, HANDLER)
```

- Donde:
 - **app** es una instancia de express
 - **METHOD** es el método de solicitud HTTP
 - **PATH** es la ruta de acceso en el servidor
 - **HANDLER** es la función que se ejecuta cuando se correlaciona la ruta

- **Cómo implementar una API REST con Express**
 - Nuevo recurso: POST
 - Devolver todos los recursos: GET
 - Devolver un recurso concreto: GET
 - Borrar un recurso: DELETE
 - Actualizar un recurso: PUT

- Servidor y módulos

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

//Generate unique id for resources
const uuid = require('uuid/v4');

const app = express();

//Convert json bodies to JavaScript object
app.use(express.json());

//Save info in memory
const ads = new Map();

// Rest methods here...

app.listen(3000, () => { console.log('Server started in port 3000') });
```

- Nuevo recurso (POST)

```
app.post('/ads', (req, res) => {  
  const ad = req.body;  
  //Validation  
  if (typeof ad.message !== 'string' || typeof ad.author !== 'string') {  
    res.sendStatus(400);  
  } else {  
    //Create object with needed fields and assign id  
    const newAd = {  
      id: uuid(),  
      message: ad.message,  
      author: ad.author  
    };  
    //Save resource  
    ads.set(newAd.id, newAd);  
    //Return new resource  
    res.json(newAd);  
  }  
});
```

- Obtener todos los recursos (GET)

```
app.get('/ads', (req, res) => {  
  const allAds = [...ads.values()];  
  res.json(allAds);  
});
```


- Obtener un recurso (GET)

```
app.get('/ads/:id', (req, res) => {  
  const id = req.params.id;  
  const ad = ads.get(id);  
  if (!ad) {  
    res.sendStatus(404);  
  } else {  
    res.json(ad);  
  }  
});
```

- **Borrar un recurso (DELETE)**

```
app.delete('/ads/:id', (req, res) => {  
  const id = req.params.id;  
  const ad = ads.get(id);  
  if (!ad) {  
    res.sendStatus(404);  
  } else {  
    ads.delete(id);  
    res.json(ad);  
  }  
});
```

- Actualizar un recurso (PUT)

```
app.put('/ads/:id', (req, res) => {  
  const id = req.params.id;  
  const ad = ads.get(id);  
  if (!ad) {  
    res.sendStatus(404);  
  } else {  
    const adReq = req.body;  
    //Validation  
    if (typeof adReq.message !== 'string' || typeof adReq.author !== 'string') {  
      res.sendStatus(400);  
    } else {  
      //Create object with needed fields and assign id  
      const newAd = {  
        id,  
        message: adReq.message,  
        author: adReq.author  
      };  
      //Update resource  
      ads.set(id, newAd);  
      //Return new resource  
      res.json(newAd);  
    }  
  }  
});
```

- Cualquier método se indica con “all”

```
app.all('/secret', (req, res) => {  
  console.log('Accessing the secret section ...');  
});
```

- Se pueden configurar varios métodos para la misma ruta y decidir si se ejecuta el siguiente o no

```
app.all('/secret', (req, res, next) => {  
  console.log('Accessing the secret section ...');  
  next(); // pass control to the next handler  
});
```

<https://expressjs.com/es/api.html#res>

- **Métodos de respuesta**

- **res.download():** Solicita un archivo para descargarlo
- **res.end():** Finaliza el proceso de respuesta
- **res.json():** Envía una respuesta JSON
- **res.jsonp():** Envía una respuesta JSON con soporte JSONP
- **res.redirect():** Redirecciona una solicitud
- **res.render():** Representa una plantilla de vista
- **res.send():** Envía una respuesta de varios tipos
- **res.sendFile():** Envía un archivo como una secuencia de bytes
- **res.sendStatus():** Establece el código de estado de la respuesta y envía su representación de serie como el cuerpo de respuesta

- Definición de varios métodos para la misma ruta

```
app.route('/book')
  .get((req, res) => {
    res.send('Get a random book');
  })
  .post((req, res) => {
    res.send('Add a book');
  })
  .put((req, res) => {
    res.send('Update the book');
  });
```

- Servir ficheros estáticos

- Configuración en código para servir los ficheros de la carpeta public por http

```
app.use(express.static('public'));
```

- Los ficheros estáticos están accesibles en la raíz

<http://localhost:3000/images/gatito.jpg>

<http://localhost:3000/hello.html>

Ejercicio 1

- Implementa una **API REST** en el servidor para gestionar **Items**
- Los items se gestionarán en **memoria** (como en el ejemplo de los anuncios)

- **API REST Items**

- **Creación de items**

- Method: POST
 - URL: `http://127.0.0.1:8080/items/`
 - Headers: Content-Type: `application/json`
 - Body:

```
{ "description" : "Leche", "checked": true }
```

- Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK)

- **API REST Items**

- **Consulta de items**

- Method: GET
 - URL: `http://127.0.0.1:8080/items/`
 - Result:

```
[  
  { "id": 1, "description": "Leche", "checked": false },  
  { "id": 2, "description": "Pan", "checked": true }  
]
```

- Status code: 200 (OK)

- **API REST Items**
 - **Consulta de item concreto**
 - Method: GET
 - URL: `http://127.0.0.1:8080/items/1`
 - Result:

```
{ "id": 1, "description": "Leche", "checked": false }
```

- Status code: 200 (OK)

- **API REST Items**

- **Modificación de items**

- Method: PUT
 - URL: `http://127.0.0.1:8080/items/1`
 - Headers: Content-Type: application/json
 - Body:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK)

- **API REST Items**

- **Borrado de items**

- Method: DELETE
 - URL: `http://127.0.0.1:8080/items/1`
 - Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

- Status code: 200 (OK)