ETSIT UPM

dit UPM

Node, npm, Express

Santiago Pavón

Versión: 2017-04-17

Repaso

- Los clientes y servidores son programas.
  - que se conectan con un socket y hablan HTTP.
- Usamos el módulo http de node para procesar las peticiones y las respuestas.

```
var http = require('http');
http.createServer(function(request, response) {
    . . .
}).listen(3000);
```

- Usaremos el módulo express para organizar el código de la aplicación servidora.
  - Middlewares

Petición y respuesta HTTP intercambiadas en el ejemplo anterior

- **Petición HTTP:**

```
GET / HTTP/1.1
Host: localhost:3000
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)
       AppleWebKit/536.11 (KHTML, like Gecko)
       Chrome/20.0.1132.57 Safari/536.11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
       */*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: es-ES,es;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

- **Respuesta HTTP:**

```
HTTP/1.1 200 OK
Content-Length: 74
Content-Type: text/html

<html><head><title>Hola Mundo</title></head>
       <body>Hola Mundo</body></html>
```

Hay que instalar módulo express:

$ npm install express

Directorio de un proyecto y paquete:
npm, package.json y node_modules

Juan Quemada, DIT - UPM

Paquete npm

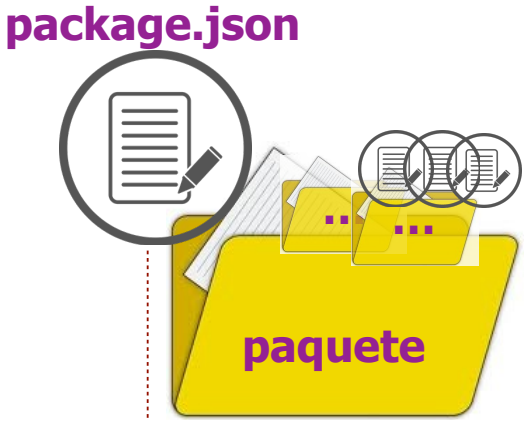
- **Paquete** (directorio)
  - Conjunto de programas que se distribuyen conjuntamente, para facilitar su
    - documentación, búsqueda, inspección, distribución, instalación, uso por terceros, …
- **npm**
  - meta-comando UNIX de gestión de paquetes
    - **npm** añade el fichero **package.json** en el directorio raíz del proyecto para gestión del paquete
  - Documentación: Doc: https://www.npmjs.org/doc/
  - Actualización de npm: https://docs.npmjs.com/getting-started/installing-node
    - **npm** se instala al instalar node.js, pero se actualiza a menudo y conviene actualizarlo
  - Instalación de npm: http://blog.npmjs.org/post/85484771375/how-to-install-npm

```
cal_2com — -bash — 70×27
venus:cal_2com jq$
venus:cal_2com jq$ npm --help

Usage: npm <command>

where <command> is one of:
    access, adduser, bin, bugs, c, cache, completion, config,
    ddp, dedupe, deprecate, dist-tag, docs, edit, explore, get,
    help, help-search, i, init, install, install-test, it, link,
    list, ln, logout, ls, outdated, owner, pack, ping, prefix,
    prune, publish, rb, rebuild, repo, restart, root,
    run-script, s, se, search, set, shrinkwrap, star, stars,
    start, stop, t, tag, team, test, tst, un, uninstall,
```

npm añade **package.json** al directorio del proyecto para poder gestionar el proyecto como un paquete fácilmente utilizable por terceros.

Fichero package.json

- Fichero **package.json**
  - Indica que el directorio es un **paquete npm**
    - Debe estar en el directorio raíz del paquete
  - Permite automatizar la gestión del **paquete npm**
    - Contiene toda la **información** del paquete en formato **JSON**
- Contenido de **package.json**
  - **Metadatos** del programa: **nombre**, **versión**, …
  - **Scripts** normalizados: arranque (start), tests (test), ….
  - **Dependencias** de instalación del programa
    - Todos los paquetes **npm** utilizados por este paquete
  - ……

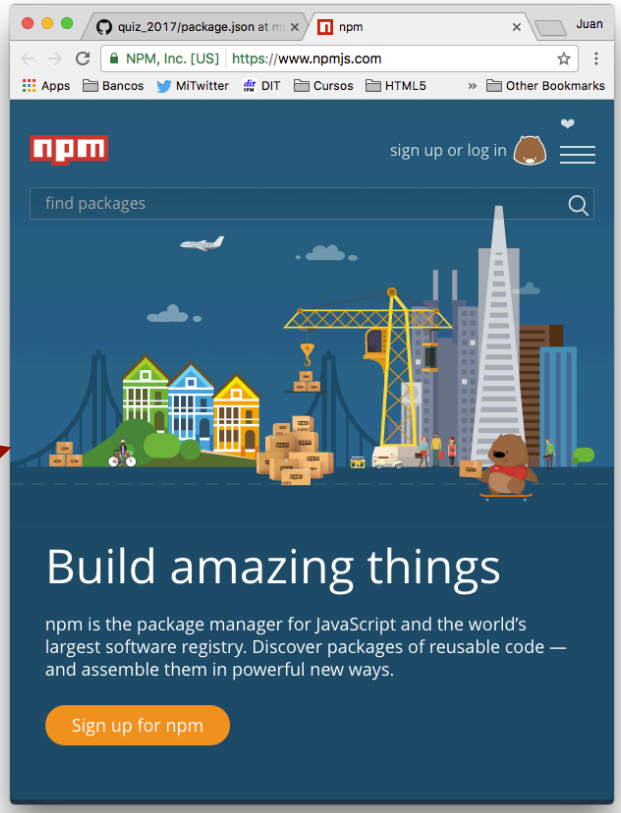


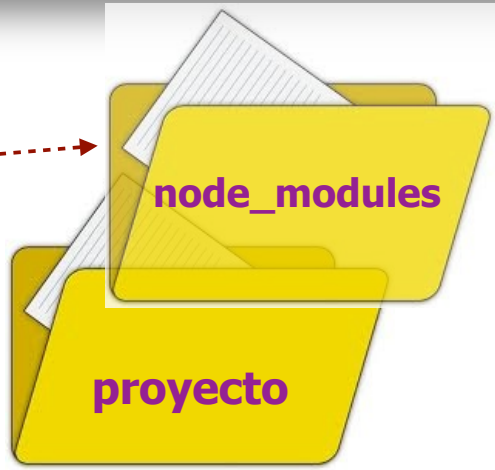

```
{
  "name": "quiz",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.13.2",
    "cookie-parser": "~1.3.5",
    "debug": "~2.2.0",
    "ejs": "~2.3.3",
    "express": "~4.13.1",
    "morgan": "~1.6.1",
    "serve-favicon": "~2.3.0"
  }
}
```

```
npm init <paquete>      // Crea paquete añadiendo fichero package.json
npm start               //  Ejecuta script start de package.json
npm test                //  Ejecuta script test de package.json (si existiese)
……..
```

Registro npm

- **Registro npm**
  - Es el repositorio central de paquetes
    - Permite **acceso Web** y con **comandos npm**
  - Está accesible en: https://www.npmjs.org
- **npm publish <paquete>**
  - **Publica** paquetes locales en el registro
    - Cualquiera podrá instalarse un paquete publicado
      - Se necesita cuenta en el registro para publicar
- **npm install <paquete>**
  - **Instala** <paquete> del registro en local
    - Se instala en directorio **node_modules** junto con sus **dependencias**
- Modelo de negocio similar a GitHub
  - Paquetes públicos se albergan gratis en el registro
  - Registros privados de pago





Módulos node.js (repaso)

- node.js incluye un sistema de **módulos**
  - Cada modulo es un **fichero diferente** con un **espacio de nombres local**
    - Documentación: http://nodejs.org/api/modules.html
- **require(<fichero>)**
  - Instrucción que importa el modulo indicado (<fichero>)
    - **<fichero>** puede ser una **ruta al directorio o fichero** concreto que contiene el módulo
    - **<fichero>** puede ser un **nombre** de modulo instalado en el directorio **node_modules**

```
var m1 = require("paquete");
var m2 = require("./paquete");
var m3 = require("./file.js");
```

- Un módulo tiene una **parte pública** (**interfaz**) y otra **privada** (**implementación**)
  - **Interfaz**: parte visible en el exterior que permite utilizar el módulo a otros
    - Los métodos se exportan individualmente con: **exports.metodo_individual = <método>**
    - El objeto interfaz completo se exporta completo con: **module.exports = <objeto_interfaz>**

```
exports.atributo1 = objeto;
exports.atributo2 = objeto;
exports = module.exports = objeto;
```

  - **Implementación**: código del módulo que crea la funcionalidad
    - El bloque de código se encapsula en un **cierre** (closure) para aislar el espacio de nombres interno

Agenda: modulo node.js que encapsula un cierre

```
module.exports = function (title, init) {
  var _title = title;
  var _content = init;

  return {
    title: function()           { return _title; },
    add:  function(nombre, tf) { _content[nombre]=tf; },
    tf:     function(nombre)    { return _content[nombre]; },
    remove: function(nombre)    { delete _content[nombre]; },
    toJSON: function()          { return JSON.stringify(_content);}
  }
}
```

Modulo: **agenda_mod_closure.js** exporta un cierre (closure) que devuelve un objeto con los métodos de acceso a las variables de la función.

var Agenda = require('./agenda_mod_closure') importa el módulo agenda_mod_closure.js y lo guarda en la variable agenda. **Nota**: los módulos JS se guardan en variables globales.

La variable **friends** almacena una instancia de la agenda inicializada con teléfonos de amigos.

La variable **work** guarda otra instancia de la agenda inicializada con teléfonos del trabajo.

```
var agenda = require('./agenda_mod_closure');

var friends = agenda ("friends",
                      { Peter: 913278561,
                        John:  957845123
                      });
friends.add("Jesus", 978512355);

var work = agenda ("Work",
                   { "Peter Tobb": 913278561,
                     "Paul Smith": 957845123
                   });

console.log('Telephone of Peter:    ' + friends.tf("Peter"));
console.log('Telephone of Jesus:    ' + friends.tf("Jesus"));
console.log('Telephone of Edith:    ' + friends.tf("Edith"));
console.log();
console.log('Tf of Peter Tobb: ' + work.tf("Peter Tobb"));
console.log('Work:    ' + work.toJSON());
```

```
venus:agenda_mod jq$
venus:agenda_mod jq$ node agenda_mod_closure_main.js
Telephone of Peter:    913278561
Telephone of Jesus:    978512355
Telephone of Edith:    undefined

Tf of Peter Tobb: 913278561
Work:    {"Peter Tobb":913278561,"Paul Smith":957845123}
venus:agenda_mod jq$
```

agenda_mod_class_main.js: **programa principal** que importa el fichero agenda_mod_closure.js (modulo), crea 2 agendas (friends, work) y muestra por consola partes de su contenido.

Express.js

¿Qué es **express**?

- Documentación:

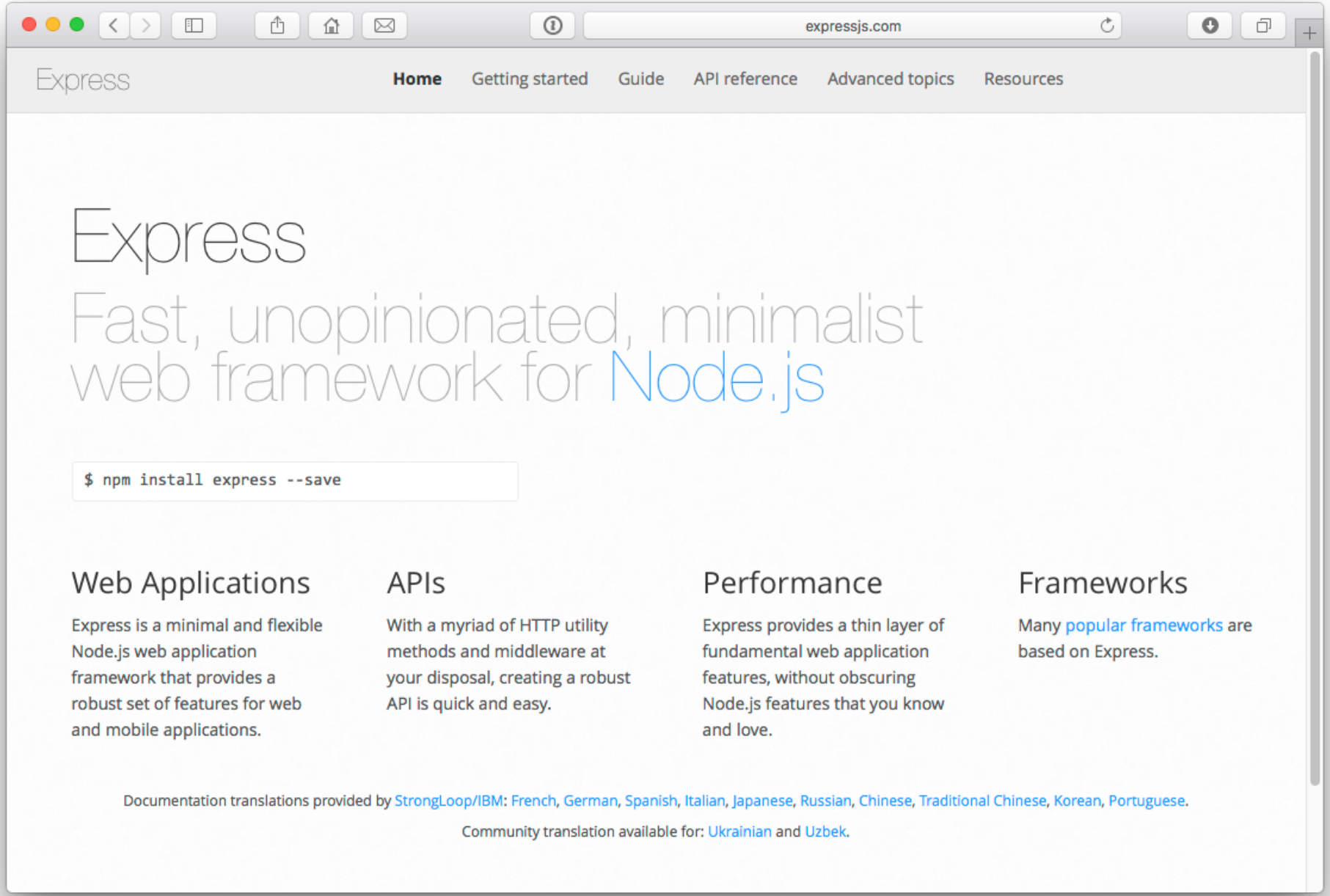
  `http://expressjs.com/guide.html`

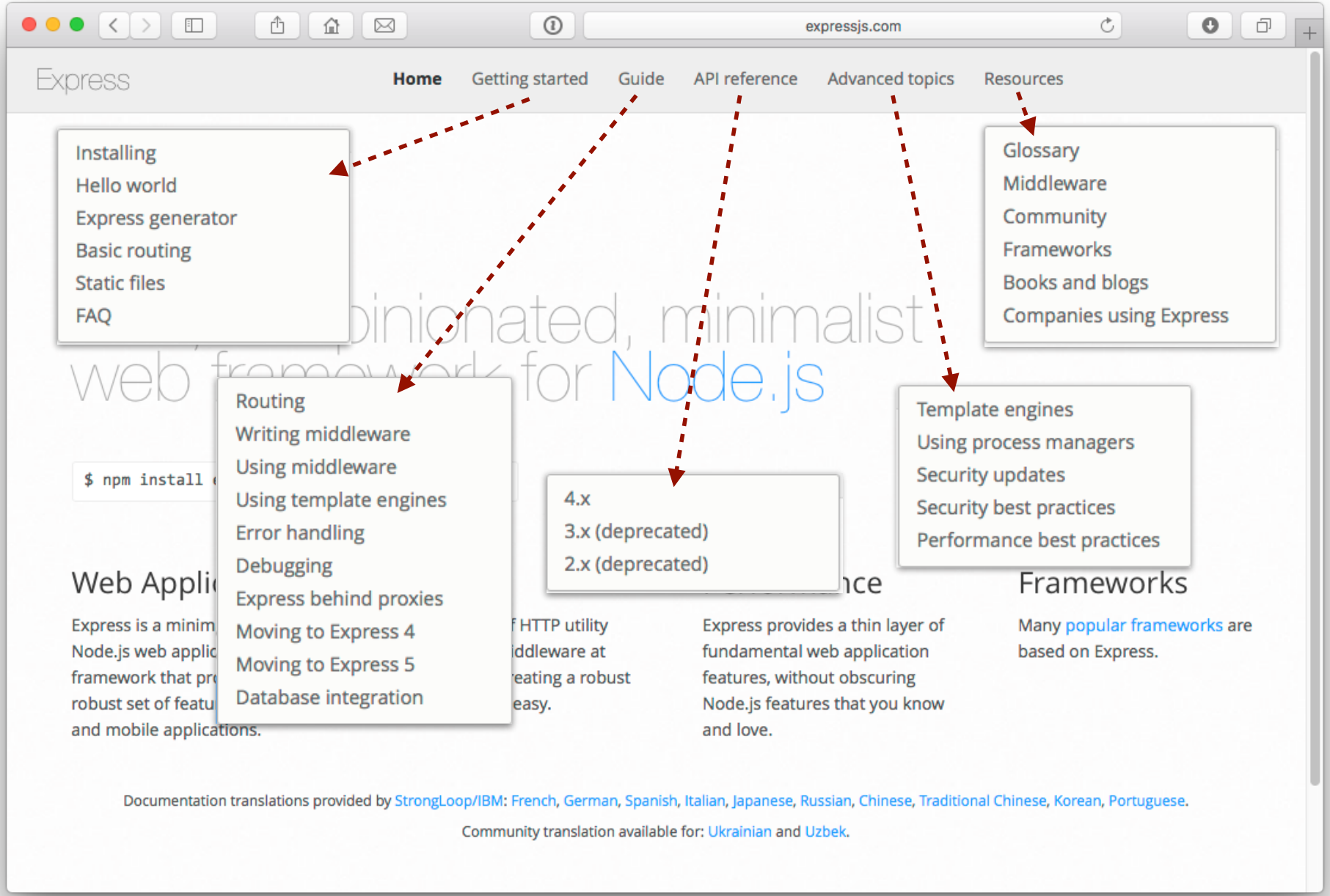
  `https://github.com/visionmedia/express`

- Es un framework para el desarrollo de aplicaciones Web con Node.js.

- Características:

  Extiende connect (uso de middlewares), manejo de **rutas**, soporte de múltiples motores de **plantillas** para la generación de vistas, negociación del **formato** de los contenidos, configurable para entornos de producción/desarrollo/pruebas, módulos adicionales para crear rápidamente una versión inicial de la aplicación, etc.





Hay que instalar módulo express:

$ npm install express

Middlewares

- Express es un framework node para programar servidores web.
  - Las funcionalidades del servidor se programan **middlewares**.
- Los middlewares son funciones con la misma interface:

```
function(request,response,next)
```

- Cada función middleware de la cadena se encarga de una tarea.
- Con los middlewares se forma una cadena de llamadas entre ellos.
  - Un middleware **pasa el control** al siguiente middleware llamando a la función **next**() que le han pasado como parámetro.
  - Un middleware **termina** la cadena de llamadas si no llama a **next**().
    - En este caso, debe proporcionar la respuesta HTTP a enviar.
  - Los parámetros **request** y **response** son referencias a los objetos que representan la petición y la respuesta HTTP en curso.
    - Los middlewares suelen añadir a estos objetos **nuevos atributos** como efecto lateral.
      - donde guardan resultados de su ejecución.

Manejo de los Errores

- La ejecución secuencial de los middlewares se interrumpe si:
  - Alguna sentencia lanza una excepción.
  - Nosotros lanzamos una excepción programáticamente:
    ```
    throw new Error('Houston, tenemos un problema');
    ```
  - Llamamos a next pasándole un parámetro:
    ```
    next(new Error('Houston, tenemos otro problema'));
    ```
- En estos casos la ejecución secuencial de middlewares salta al siguiente middleware de manejo de errores.
  - Son los que tienen la signatura: `function(error, req, res, next);`
- El middleware de manejo de errores:
  - Seguramente llamará a res.writeHead, res.write y res.end para contestar él mismo.
  - Si arregla el problema llamará a next() para seguir con el siguiente middleware.
  - O puede que lance otra excepción o llame a next(error).
    - Se saltará al siguiente middleware de manejo de errores.

Uso

- Crear una **Aplicación Express**:
  - Se crea con la función express()

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

var app = express();
```

- **Instalar Middlewares:**
  - Se instalan con:

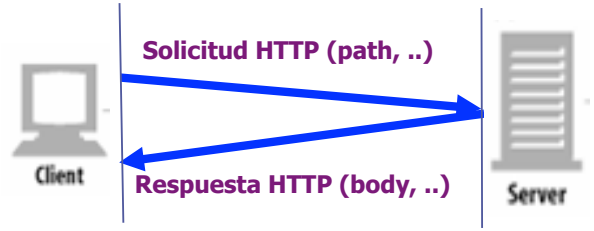
```
app.use(<middleware1>);

app.use(<middleware2>);

app.use(<middleware3>, <middleware4>);

app.use(<path>, <middleware3>);
```

  - Se ejecutan en el mismo orden en que han sido instalados.
  - Se ejecutan cada vez que llega una petición HTTP.
  - Documentación: http://expressjs.com/4x/api.html#app.use

18

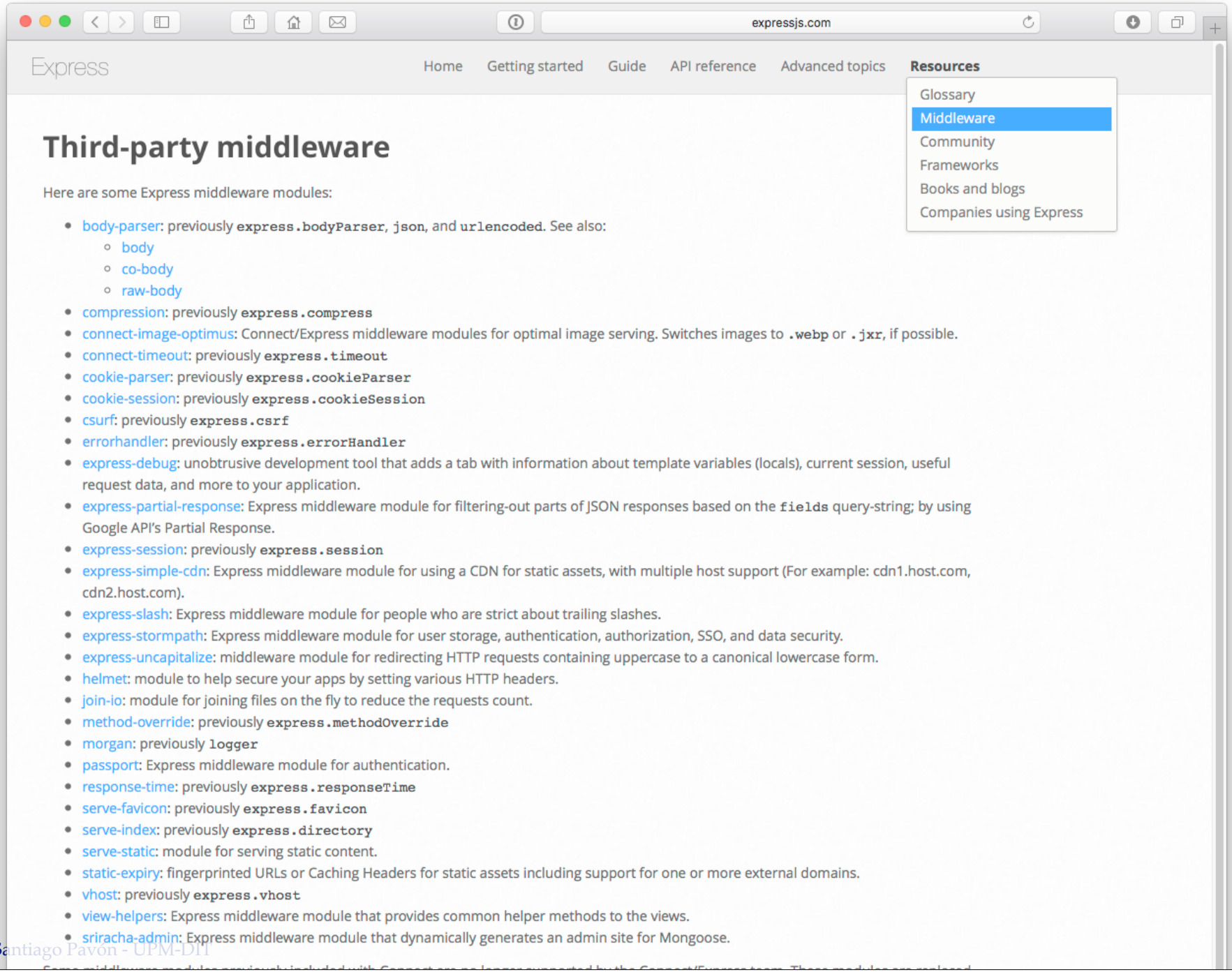
Rutas

express
web application
framework for
node



- Instalar middlewares asociándolos con:
  - un path y un método HTTP: GET, PUT, POST, DELETE.
    - MW solo se ejecuta si coinciden el método y el path con la solicitud HTTP.
- Se instalan usando:

```
var router = express.Router();

router.get(path, MW);
router.post(path, MW);
router.put(path, MW);
router.delete(path, MW);
router.all(path, MW);
```



Motor de Vistas EJS

- **EJS** = Javascript embebido.
- Los ficheros de vistas EJS contienen:
  - texto HTML.
  - código javascript entre las marcas `<%` y `%>`.
  - expresiones javascript entre las marcas `<%=` y `%>`.
    - El valor de las expresiones se incorpora al texto HTML.
    - Previamente se escapan los caracteres conflictivos para evitar inyección de código:
      - `<` se sustituye por `<`
      - `>` se sustituye por `>`
      - `&` se sustituye por `&`
      - ...
  - expresiones javascript entre las marcas `<%-` y `%>`.
    - El valor de las expresiones se incorpora al texto HTML.
    - No se escapan los caracteres conflictivos.
  - inclusión de ficheros con `<% include` *path_del_fichero_a_incluir* `%>`.
- Documentación: `https://github.com/visionmedia/ejs`

Ejemplo: Quiz

- En el proyecto quiz:

  `https://github.com/CORE-UPM/quiz_2017.git`

- Ver los siguiente ficheros:

  `app.js`

  `controllers/*`

  `routes/index.js`

  `views/*`

express-generator

- Lo primero es instalar el módulo express-generator:

```
$ npm install express-generator
```

  - *Los paquetes se instalan en el subdirectorio ~/node_modules.*
  - *Para instalarlos a nivel de sistema (global) pasar la opcion -g.*
    - *Se instalarán en /usr/local/lib/node_modules.*

- Crear Esqueleto de una Aplicación:

```
$ express demo // crea ficheros iniciales (Instalado globalmente)
$ ./node_modules/.bin/express demo // crea ficheros iniciales (inst local)
$ cd demo
$ npm install  // instala las dependencias declaradas en package.json
```

- Ejecutamos la aplicación:

```
$ node bin/www
```

- Y nos conectamos con un navegador a `http://localhost:3000`.