Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Práctica 2
Node.js y publicación de servicios

Sistemas y Tecnologías Web

Grado en Ingeniería Informática

Curso 2024-2025

Francisco Javier Fabra Caro

jfabra@unizar.es

Gestión de versiones

Gestión de versiones

- O cómo tener diferentes versiones de Node.js corriendo en la misma máquina
  - Si no tienes **docker** (que facilitaría las cosas)
- Dos alternativas principales:
  - **n**
    - https://github.com/tj/n
  - **Node Version manager**
    - https://github.com/nvm-sh/nvm
  - Lectura recomendada:

    https://www.sitepoint.com/quick-tip-multiple-versions-node-nvm/

NPM

Node Package Manager

- NPM (v11.1.0, Enero de 2025)
  - Propiedad de GitHub desde Marzo de 2020
    - https://github.blog/2020-03-16-npm-is-joining-github/
  - Permite buscar paquetes (***prebuilt***) en el repositorio oficial https://www.npmjs.com
  - Permite instalar paquetes, gestionar las versiones y gestionar las dependencias entre paquetes en NodeJS
  - Objetivo: extender la funcionalidad de la aplicación
  - Actualmente, hay más de3,1Mill de paquetes disponibles (feb 2025)
    - En 2015/16 había unos 46.000
    - En 2018, más de 350.000
    - En 2019 se superó el millón de paquetes
    - https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html

package.json

- Fichero que se aloja en el raíz del proyecto
- Contiene metadatos del proyecto
- Incluye las dependencias

```
1  {
2    "name": "application-name",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "body-parser": "~1.15.2",
10     "cookie-parser": "~1.4.3",
11     "debug": "~2.2.0",
12     "express": "^4.14.0",
13     "morgan": "^1.7.0",
14     "pug": "^2.0.0-beta6"   ,
15     "serve-favicon": "~2.3.0"
16   }
17 }
```

1

2

package.json

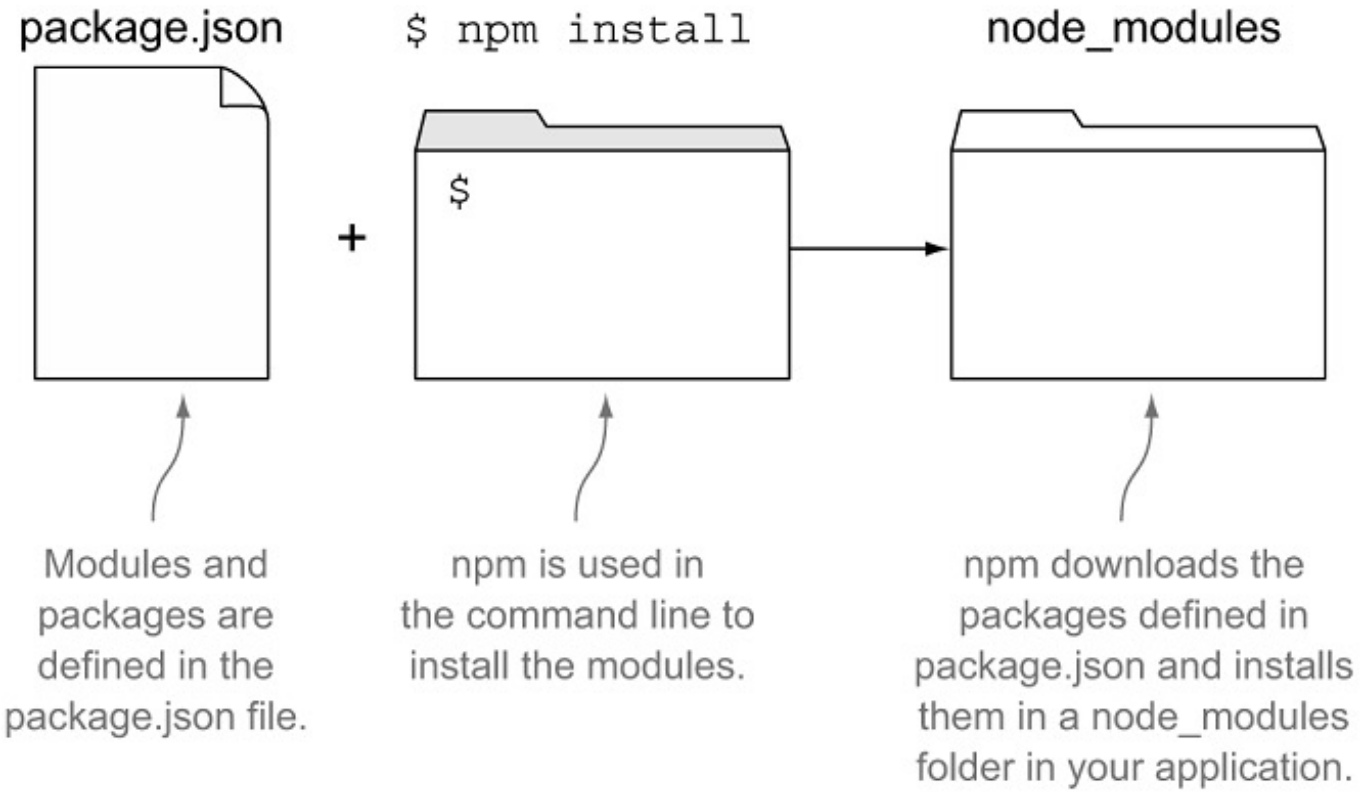
- *Express 4.14.0*
  - Major version (4)
  - Minor version (14)
  - Patch version (0)
- Prefijo ”~”
  - *Latest patch version available*
- Prefijo “^”
  - *Latest minor version available*
- Buena práctica: no tocar la *major* version

```
 8   "dependencies": {
 9     "body-parser": "~1.15.2",
10     "cookie-parser": "~1.4.3",
11     "debug": "~2.2.0",
12     "express": "^4.14.0",
13     "morgan": "^1.7.0",
14     "pug": "^2.0.0-beta6"   ,
15     "serve-favicon": "~2.3.0"
16   }
17 }
```

Comandos básicos

```
$ npm --version

$ sudo npm install npm -g (--global)

$ npm ls [-g]

$ npm search <módulo>

$ npm install <módulo> [-g]

$ npm uninstall <módulo>

$ npm update <módulo>
```

Comandos básicos



Más comandos

Añadir paquetes a un proyecto existente

```
$ npm install --save package-name
```

Actualizar los paquetes a la última versión

```
$ npm install
```

Más comandos

```
$ npm config list

$ npm config get prefix

      $ cd && mkdir .node_modules_global
      $ npm config set prefix=$HOME/.node_modules_global

$ npm list --global

$ npm list -g --depth=0

$ npm init
$ npm init --yes
```

package.json


```
{
   "name": "express",
      "description": "Fast, unopinionated, minimalist web framework",
      "version": "4.11.2",
      "author": {

         "name": "TJ Holowaychuk",
         "email": "tj@vision-media.ca"
      },

   "contributors": [{
      "name": "Aaron Heckmann",
      "email": "aaron.heckmann+github@gmail.com"
   },

   {
      "name": "Young Jae Sim",
      "email": "hanul@hanul.me"
   } ],
   "license": "MIT", "repository": {
      "type": "git",
      "url": "https://github.com/strongloop/express"
   },
   "homepage": "https://expressjs.com/", "keywords": [
      "express",
      "framework",
      "api"
   ],
   "dependencies": {
      "accepts": "~1.2.3",
      "content-disposition": "0.5.0",
      "cookie-signature": "1.0.5",
      "debug": "~2.1.1",
      "depd": "~1.0.0",
      "utils-merge": "1.0.0"
   },
   "devDependencies": {
      "after": "0.8.1",
      "ejs": "2.1.4",
      "istanbul": "0.3.5",
      "marked": "0.3.3",
      "vhost": "~3.0.0"
   },
   "engines": {
      "node": ">= 0.10.0"
   },
   "files": [
      "LICENSE",
      "History.md",
      "Readme.md",
      "index.js",
      "lib/"
   ],
   "scripts": {
      "test": "mocha --require test/support/env
         --reporter spec --bail --check-leaks test/ test/acceptance/",
      "test-cov": "istanbul cover node_modules/mocha/bin/_mocha
         -- --require test/support/env --reporter dot --check-leaks test/ tes
      "test-tap": "mocha --require test/support/env
         --reporter tap --check-leaks test/ test/acceptance/",
      "test-travis": "istanbul cover node_modules/mocha/bin/_mocha
         --report lcovonly -- --require test/support/env
         --reporter spec --check-leaks test/ test/acceptance/"
   },
   "gitHead": "63ab25579bda70b4927a179b580a9c580b6c7ada",
   "bugs": {
      "url": "https://github.com/strongloop/express/issues"
   },
   "_id": "express@4.11.2",
   "_shasum": "8df3d5a9ac848585f00a0777601823faecd3b148",
   "_from": "express@*",
   "_npmVersion": "1.4.28",
   "_npmUser": {
      "name": "dougwilson",
      "email": "doug@somethingdoug.com"
   },
   "maintainers": [
      {
         "name": "tjholowaychuk",
         "email": "tj@vision-media.ca"
      },
      {
         "name": "rfeng",
         "email": "enjoyjava@gmail.com"
      }
   ],
   "dist": {
      "shasum": "8df3d5a9ac848585f00a0777601823faecd3b148",
      "tarball": "https://registry.npmjs.org/express/-/express-4.11.2.tgz"
   },
   "directories": {},
      "_resolved": "https://registry.npmjs.org/express/-/express-4.11.2.tgz",
      "readme": "ERROR: No README data found!"
}
```

Módulos de apoyo

Manejo de las subidas

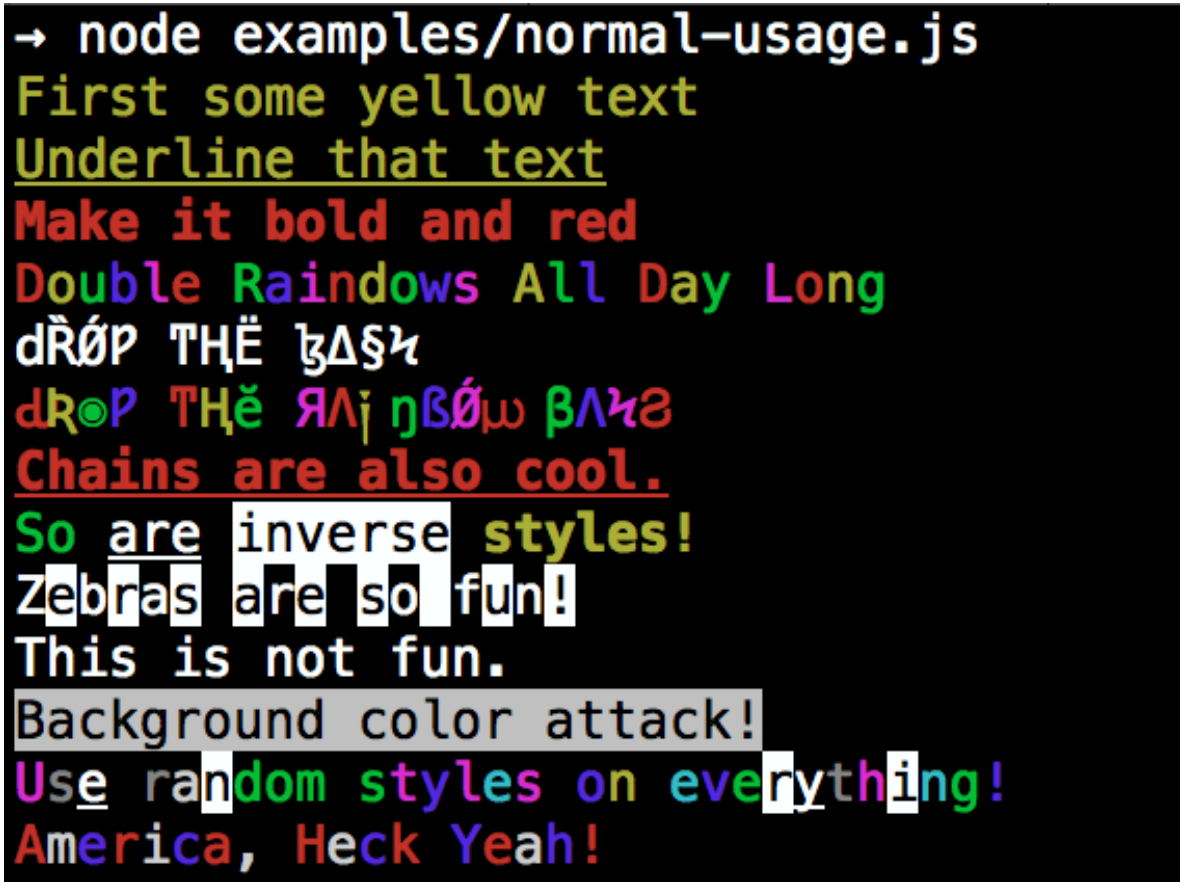
```
$ npm install formidable
..
npm info build Success: formidable@1.0.17
npm ok
```

- Facilita el procesamiento de formularios enviados vía HTTP POST
  - Tenemos que crear un *IncomingForm*
  - Abstracción del formulario enviado
  - Lo utilizaremos para parsear el objeto *request* al servidor HTTP

https://www.npmjs.com/package/formidable

Colors

https://www.npmjs.com/package/colors



fancy-log

https://www.npmjs.com/package/fancy-log

```
var log = require('fancy-log');

log('a message');
// [16:27:02] a message

log.error('oh no!');
// [16:27:02] oh no!
```

Debugging

- Básicamente:
  - `console.log('some msg')`
  - `console.error('some error')`

- Pero hay cosas más *elegantes*:
  - Debug
  - **Winston** / Morgan
    - Winston: Multi-transport async logging library for Node.js
    - https://github.com/winstonjs/winston

Paquete debug

```
$ npm install debug --save
```

```
const debug = require('debug')('my-namespace')
const name = 'my-app'
debug('booting %s', name)
```

```
$ DEBUG=my-namespace node app.js
```

Paquete debug

```
$ DEBUG=my-namespace,express* node app.js
```

```
express:application set "x-powered-by" to true +0ms
express:application set "etag" to 'weak' +3ms
express:application set "etag fn" to [Function: wetag] +2ms
express:application set "env" to 'development' +1ms
express:application set "query parser" to 'extended' +0ms
express:application set "query parser fn" to [Function: parseExtendedQueryString] +0ms
express:application set "subdomain offset" to 2 +0ms
express:application set "trust proxy" to false +0ms
express:application set "trust proxy fn" to [Function: trustNone] +1ms
express:application booting in development mode +0ms
express:application set "view" to [Function: View] +0ms
express:application set "views" to '/Users/gergelyke/Development/risingstack/trace/heapdump-experiment/views' +1ms
express:application set "jsonp callback name" to 'callback' +0ms
express:router use / query +1ms
express:router:layer new / +0ms
express:router use / expressInit +1ms
express:router:layer new / +0ms
express:router:route new / +0ms
express:router:layer new / +0ms
express:router:route get / +1ms
express:router:layer new / +0ms
express:router dispatching GET / +11s
express:router query  : / +2ms
express:router expressInit  : / +1ms
express:router dispatching GET /favicon.ico +249ms
express:router query  : /favicon.ico +1ms
express:router expressInit  : /favicon.ico +0ms
finalhandler default 404 +1ms
```

Publicación de servicios con Express

Pasos a seguir en la creación de un API REST para la aplicación

1. Creación de una aplicación Express
2. Gestión de las dependencias de la aplicación con npm
3. Adaptar Express para la arquitectura MVC
4. Publicar servicios con express
5. Documentar servicios con Swagger
6. Validar servicios con Swagger, Postman u otra herramienta similar

Publicación de servicios

```
router.get('/test', function(req, res) {
    res.send("<h3>GET over test/</h3>");
});

router.post('/test', function(req, res) {
    console.log(req.body);
    // console.log(JSON.stringify(req.body, null, 4));
    res.send("<h3>POST over test/</h3>");
});

router.delete('/test', function(req, res) {
    res.status(400);
    res.send('No podrás pasar!');
});
```

Ejercicio

- Implementación del API de estudiantes.
- Especificación disponible en Moodle.
  - `students.js`
- Fichero con implementación completa.

Validación y testing de servicios

Validación y *testing* de servicios

- Navegador
- Curl
- Postman
  - http://postman.com
- Insomnia
  - https://insomnia.rest/
- Y más..

Validación de mensajes

Validación del JSON del mensaje

- ¿Cómo podemos verificar que lo que me que llega en el body en las peticiones POST y PUT es válido?
  - **JSON Schema**
  - Módulo **ajv**
  - *Ver students-schema.js*

Documentación de servicios

Swagger

- http://swagger.io/
- El objetivo de Swagger es definir una interfaz para las API REST que permita descubrir y entender las capacidades de un servicio
  - Sin necesidad de acceder al código fuente
  - Sin documentación adicional
  - Sin inspección del tráfico de red
- Swagger Specification: http://swagger.io/specification/
- El consumidor puede entender e interactuar con el servicio remoto con la mínima implementación y sin conocer detalles técnicos de las llamadas
- Cuando hagamos cambios sobre el código, la documentación se actualizará y sincronizará con el API automáticamente

http://swagger.io/getting-started-with-swagger-i-what-is-swagger/

Enfoques para desarrollar el API

- **Top-down**
  1. Utilizamos el **Swagger Editor** para crear una definición de los servicios que implementaremos
     - http://editor.swagger.io
     - La especificación se escribe en YAML
     - Se puede previsualizar la documentación en Swagger *on-the-fly*
  2. Usamos las herramientas **Swagger Codegen** para generar la implementación del servidor
     - https://swagger.io/tools/swagger-codegen/
     - Soporta prácticamente cualquier lenguaje moderno de desarrollo Web

Enfoques para desarrollar el API

- **Bottom-up**
  1. Partimos de un API REST para el que queremos crear la definición Swagger
  2. A partir de aquí, tenemos dos opciones:
     a) Creamos la definición manualmente, utilizando el **Swagger Editor**
     b) Si hemos desarrollado la aplicación y el API REST con uno de los frameworks soportados (Node.js o JAX-RS, por ejemplo), podemos obtener la definición de Swagger **automáticamente**
     c) Si usáis JAX-RS: https://github.com/swagger-api/swagger-core/wiki/Swagger-Core-JAX-RS-Project-Setup-1.5.X

Especificación de la documentación de Swagger

https://swagger.io/specification/

swagger-ui-express + swagger-jsdoc

1. Instalar dependencias:

```
npm install –save swagger-ui-express swagger-jsdoc
```

2. Configurar Swagger JSDoc y documentar las rutas

```
const express = require('express');
const swaggerUi = require('swagger-ui-express');
const swaggerJSDoc = require('swagger-jsdoc');

const app = express();

// Opciones de configuración de Swagger
const swaggerOptions = {
  definition: {
    openapi: '3.0.0', // Versión de OpenAPI
    info: {
      title: 'API de Estudiantes',
      version: '1.0.0',
      description: 'Una API sencilla para gestionar estudiantes',
    },
  },
  // Ruta a los archivos donde se documentará la API
  apis: [‘./app_server/routes/*.js'],
};

const swaggerSpec = swaggerJSDoc(swaggerOptions);

// Sirve la documentación generada en /docs
app.use('/docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));
```

Documentar cada servicio

```
/**
 * @swagger
 * /students:
 *   get:
 *     summary: Devuelve una lista de estudiantes
 *     responses:
 *       200:
 *         description: Una lista de estudiantes
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 type: object
 *                 properties:
 *                   nombre:
 *                     type: string
 *                     description: Nombre del estudiante.
 *                   apellidos:
 *                     type: string
 *                     description: Apellidos del estudiante.
 *                   nip:
 *                     type: number
 *                     description: NIP del estudiante.
 *                   email:
 *                     type: string
 *                     description: Email del estudiante.
 */
router.get('/', function(req, res, next) { ... }
```

Schemas en swagger

¿Y si queremos usar *schemas* en la documentación?

Ver el fichero de ejemplo de documentación completa (students-swagger.js)

Ejercicio

Ejercicio

- Queremos añadir las asignaturas que un estudiante ha superado, con el nombre y calificación de la asignatura:

```
{
   "nombre": "John",
   "apellidos": "Doe",
   "nip": 783265,
   "email": "783265@unizar.es"
   "asignaturas": [
      {
         "nombre": "STW",
         "calificacion": 9.5
      }
      ]
}
```

Ejercicio

- Por seguridad, las asignaturas y sus calificaciones sólo se podrán añadir y consultar, pero no modificar ni borrar.

```
GET /api/students/:id/subjects
POST /api/students/:id/subjects
```

Payload: JSON con la asignatura y calificación

- **TAREA: añade estos dos métodos al API, completa el schema y documéntalos con Swagger.**
- **¿Qué ocurre con el PUT y el DELETE?**

Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Práctica 2
Node.js y publicación de servicios

Sistemas y Tecnologías Web

Grado en Ingeniería Informática

Curso 2024-2025

Francisco Javier Fabra Caro

jfabra@unizar.es