
Academy_
FULL STACK
DEVELOPER
BOOTCAMP
Desarrollamos
{ talento }

GEEKSHUBS Academy_

Express

Qué es Express

Express es un framework para crear aplicaciones web en Node.js

Proporciona un conjunto sólido de características para las aplicaciones web y móviles.

- Web oficial

Usos

- Crear un API rest:
  - Interacción con la base de datos
  - Servir JSON a un cliente.
- Crear un backend que sirva HTML:
  - Sirve HTML estático
  - o Sirve HTML dinámico con un motor de plantillas (cómo: handlebars , pug o ejs)

Instalando Express

Debemos crear nuestro package.json desde la consola.

```
$ npm init
```

Una vez creado el package.json, instalamos express

```
$ npm install express --save
```

Otras herramientas

Una herramienta muy util y que incorpora Express es Express Generator nos automatiza y nos crea un esqueleto tipo para una app en Express

Para instalarlo debemos ejecutar:

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

GEEKSHUBS Academy_

PRIMEROS PASOS CON EXPRESS

Utilizando Express

Cargamos el objeto express en nuestro proyecto y lo inicializamos.

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

Utilizando Express

Instanciamos el servidor.

```
app.listen(3000,()=>console.log('Servidor levantado en 3000'));
```

Utilizando Express

Routing con express

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

Uilizando Express

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

app.get('/',(req, res) => res.send('Hello World!'));
app.get('/about',(req, res)=> res.send("Quienes somos"));

app.listen(3000,()=>console.log('Servidor levantado en 3000'));
```

Utilizando Express

Express podemos utilizarlo para enviar archivos de todo tipo directamente como respuesta, como por ejemplo nuestro index.html:

```
app.get('/index', (req, res) => {
    res.sendFile(`${__dirname}/public/index.html`);
});
```

GEEKSHUBS Academy_

Creando nuestra primera aplicación.

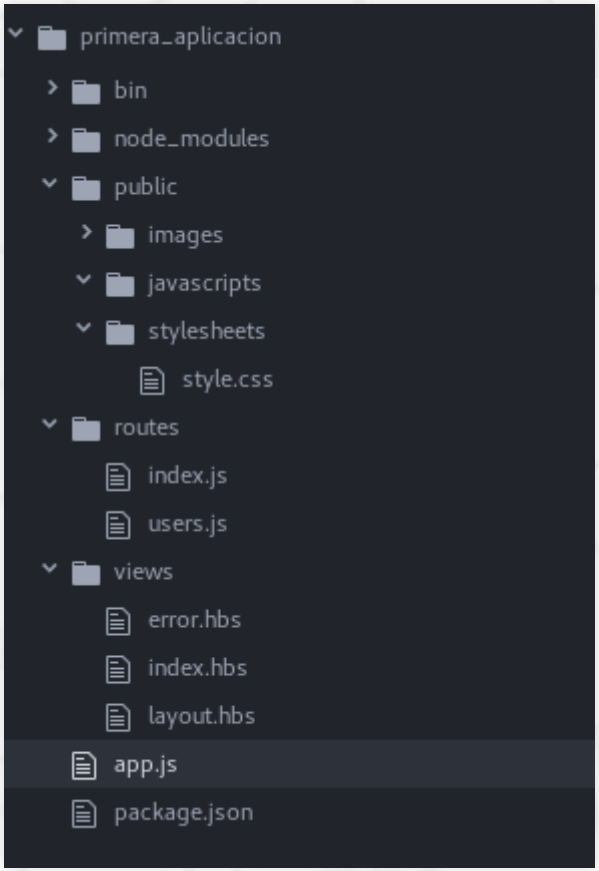
Creando nuestro primer proyecto

Vamos a crear nuestra primera aplicación utilizando Express Generator

```
$ express --view=hbs primera-aplicacion
```

Creando nuestro primer proyecto

Vemos que nos ha creado el proyecto una estructura de directorios



Creando nuestro primer proyecto

Para arrancar el proyecto primero debemos ejecutar la instalación de las dependencias

```
$ npm install
```

Una vez instalado las dependencias sólo debemos ejecutar

```
$ npm start
```

Creando nuestro primer proyecto

Para abrir nuestra web sólo debemos abrir el navegador e iniciar un localhost: 3000

Creando nuestro primer proyecto

Para poder actualizar en tiempo de ejecución los achivos del servidor utilizaremos la librería nodemon para instarlo ejecutaremos

```
$ npm install nodemon --save-dev
```

Creando nuestro primer proyecto

Ahora sólo modificamos el package.json para ejecutarlo cuando hagamos el **npm start**

```
"scripts": {
    "start": "nodemon ./bin/www"
}
```

Creando nuestro primer proyecto

Creando nuestro primera template

Para ello en el directorio de **views** creamos un nuevo archivo denominado template.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```

Creando nuestro primer proyecto

Creando nuestra primera vista con el template recién creado para ello creamos la vista en **views** denominado vista.hbs y añadimmos lo siguiente:

```
{{ page }}
```

Creando nuestro primer proyecto

Para poder enlazar las páginas debemos ahora seleccionar la ruta en el archivo index.js en la carpeta routes y añadir el siguiente código:

```
router.get('/inicio', (req, res, next) =>{
    res.render('vista.hbs',
    {
      title:'1° Vista',
      page: 'Página variable',
      layout: 'template.hbs'
    });
});
```

Creando nuestro primer proyecto

Tal y como podemos observar los siguientes elementos se establecen:

- title -> Título de la página
- page -> Información con la variable que se pasa a la vista.
- layout -> Corresponde al template que se utiliza como base de nuestras vistas.

GEEKSHUBS Academy_

APRENDIENDO HANDLEBARS

Utilizando HandleBars

HandleBars en un motor de plantillas para construir **Plantillas Semánticas**

Tenemos información sobre el mismo en https://handelbarsjs.com

Utilizando HandleBars

HandelBars contiene helpers y funciones que nos ayudan a crear cierta lógica en el Front de la vista

Utilizando HandleBars

Vamos a crear una nueva ruta y añadimos la siguiente informacion:

```
router.get('/prueba',(req,res,next)=>{
 res.render('prueba.hbs',
 {
   usuarios: [
     {id: 1 , name: 'xavi'},
     {id: 2 , name: 'pepe'},
     {id: 3 , name: 'jesus'}
   ],
   administrador:{
     nombre: 'Xavi',
     apellidos : 'Rodriguez'
   },
   appName: 'Prueba',
   layout: 'template'
 })
});
```

Utilizando HandleBars

Creamos nuestra vista **prueba.hbs** y vamos a añadir automáticamente todos los usuarios

```
{{ appName }}
{{#if usuarios.length}}
    {{#each usuarios}}
        {{id}}: {{name}},
    {{/each}}
{{else}}
    No hay usuarios.
{{/if}}
```

Utilizando HandleBars

Podemos detectar automáticamente si un objeto esta vacío o creado con la propiedad **unless**

```
{{#unless usuarios}}
  No hay usuarios
{{/unless}}
```

Utilizando HandleBars

Podemos carga un objeto en partícular.

```
{{#with administrador}}
  Nombre: {{ nombre}} , Apellidos : {{ apellidos }}
{{else}}
    No existe el administrador
{{/with}}
```

Utilizando HandleBars

A la vez podemos enviar información al terminal para poder debugear.

```
<!-- aparece en la consola de node js -->
{{log "Prueba del Log"}}
```

Utilizando HandleBars

Podemos utilizar elementos parciales para nuestra web mediante handlebars, para ello debemos registrar en nuestro directorio de "partials" en **app.js**

```
var hbs = require('hbs');
hbs.registerPartials(`${__dirname}/views/partials`);
```

Utilizando HandleBars

Ahora debemos registrar en la plantilla el nuevo elemento parcial, que debe corresponderse con el nombre del archivo, por ejemplo "partials.hbs"

```
{{> partial }}
```

Utilizando HandleBars

Ahora creamos el directorio y el nuevo archivo dentro de el denominado **partials.hbs**

Y posteriormente apagamos e iniciamos de nuevo el servidor.

Utilizando HandleBars

Ahora en el nuevo archivo Partials añadimos cualquier información y a la vez se puede utilizar las propiedades de **HandleBars** sobre el

```
# Esto es un parcial
```

Utilizando HandleBars

Para tener que evitar cada vez que cambiamos un partial apagar e iniciar de nuevo el servidor utilizaremos la librería hbs-utils

```
$ npm install hbs-utils --save
```

Utilizando HandleBars

Ahora añadimos esta nueva librería a nuestra aplicación dentro de **app.js**

```
var hbs = require('hbs');
var hbsUtils = require('hbs-utils')(hbs);
hbsUtils.registerPartials(`${__dirname}/views/partials`);
hbsUtils.registerWatchedPartials(`${__dirname}/views/partials`);
```

GEEKSHUBS Academy_

Utilizando Bower como gestor de dependencias

Utilizando Bower

Una vez comprobado que tenemos Bower instalado debemos crear un archivo denominado **.bowerrc**

```
npm install bower -g
```

Utilizando Bower

De esta manera podemos forzar la descarga de todas las dependencias de Bower

```
{
  "directory": "public/components"
}
```

Utilizando Bower

Ahora debemos mapear la zona de descarga para que express sea capaz de servir la misma para ello en app.js añadimos la siguiente línea, debemos asegurarnos que el directorio esté creado

```
app.use('/components', express.static(`${__dirname}/public/components`));
```

Utilizando Bower

Una vez definido y mapeado el directorio iniciamos **Bower** mediante:

```
$ bower init
```

Y aceptamos todos comandos, eremos que nos has creado un **bower.json**

Utilizando Bower

A partir de este momento ya podemos utilizar Bower para descargar cualquier dependencia, vamos a descargar Bootstrap

```
$ bower i -S bootstrap
```

Veremos que nos ha descargado la dependencia dentro de la carpeta definida

Utilizando Bower

Ahora ya sólo debemos añadir nuestra dependencia en nuestro template

```
<link rel="stylesheet" href="components/bootstrap/dist/css/bootstrap.css">
```

GEEKSHUBS Academy_

Utilizando MYSQL

Utilizando MYSQl

Para poder utilizar MySQL primero debemos instalar el módulo msyql para ello ejecutamos

```
$ npm install --save mysql
$ npm install --save mysql2
```

Utilizando MYSQL

Aquí tenemos toda la información correspondiente al uso del módulo de mysql

!! Si usamos XAMPP: Cambiar en el archivo php.ini "upload_max_filesize" a 6M para que permita importar la base de datos "sakila", el límite por defecto son 2M.

Utilizando MYSQL

Ahora vamos a crear las conexiones suficientes para poder utilizar este nuevo módulo

Para ello crearemos una nueva carpeta denominada connection y dentro de ella un archivo denminada mysqlconnection.js

Utilizando MYSQL

Ahora vamos a crear nuestra primera instancia para la conexión a la base de datos

```
const MYSQL = require('mysql');
const CONN = MYSQL.createConnection({
  host:'localhost',
  user:'root',
  password:'44865710',
  database:'sakila'
})
module.exports = CONN;
```

Utilizando MYSQL

Ahora vamos a crear nuestro primer modelo que será un entidad que corresponde a una tabla para ello vamos a crear una nueva carpeta que se denomine models y vamos a crear un archivo denominado filmsModel.js

Utilizando MYSQL

Ahora creamos este nuevo módelo de films para sakila

```
let conn = require('../connections/mysqlconnection');
let Films ={};
Films.fetchAll = (cb)=>{
  if(!conn)  return cb("No se ha podido crear la conexión");
  const SQL = "SELECT  * FROM film LIMIT 5;";
  conn.query(SQL, (error, rows)=>{
    if (error)  return cb(error);
    else return cb(null, rows);

  })
}
module.exports = Films;
```

Utilizando MYSQL

Ahora vamos a pasar la información a la vista para ello vamos a crear una ruta que ejecuta la consulta del modelo peliculas

```
router.get('/listado',(req,res,next)=>{
  filmModel.fetchAll((error, films)=>{
    if(error) return res.status(500).json(error);
    res.render('film-list',{
      title: 'Listado de peliculas',
      layout: 'layout.hbs',
      films
    })
  })
})
```

Utilizando MYSQL

Ahora creamos la vista que mostrará la información denominada **film-list**

```
{{#if films.length}}
  {{#each films}}


          {{title}}


        {{ film_id }}, {{ release_year}}, {{ description }}


  {{/each}}
{{/if}}
```

Utilizando MYSQL

Ahora vamos a crear los métodos para insertar nuevos registros, en el modelo film

```
Films.insert = (film,cb)=>{
  if(!conn) return cb("No se ha podido crear la conexión");
  conn.query('INSERT INTO film SET ?', [film], (error, result)=>{
    if (error) return cb(error);
    return cb(null, result.insertId);
  });
}
```

Utilizando MYSQL

Ahora creamos una nueva ruta para ejecutar la inserción

```
router.get('/insertar',(req, res, next)=>{
  const FILM ={
    "title" :"Es una prueba2",
    "language_id": 1,
  };
  filmModel.insert(FILM,(error, insertID)=>{
     if (insertID){
        return res.status(200).send('Añadido pelicula ->'+ insertID);
     }
    res.status(500).json('Error guardando película ' + error);
  });
});
```

GEEKSHUBS Academy_

Utilizando EXPRESS-SESSIONS

GEEKSHUBS Academy_

EXPRESS-SESSIONS

El modulo nos permite establecer y crear una sesión de usuario donde podemos almacenar información que posteriormente podemos utilizar para gestionar la sesión.

EXPRESS-SESSIONS

Como cualquier módulo podemos instalarlo de siguiente manera express-session

```
npm install express-session --save
```

EXPRESS-SESSIONS

Ahora debemos añadir la gestión de usuarios para ello cargarmos la librería y generamos una nueva ruta

```
//Carga de libreria
var session = require('express-session');
//Carga de la ruta para admins
var admins = require('./routes/admins');
//Estableciendo las nuevas rutas.
app.use('/admins',admins);
```

EXPRESS-SESSIONS

Vamos a establecer las condiciones para nuestra sesión.

```
//Gestión de la sesión.
app.use(session({
    // Clave con la que se va a firmar el ID de las cookies
    secret: 'clavesecreta',
    // Nombre de la cookie
    name: 'super-secret-cookie-name',
    // Si se debe reguardar el objeto completo o no en cada petición.
    resave: true,
    // Si la sesión se debe guardar al crearla aunque no la modifiquemos.
    saveUninitialized: true
}));
```

EXPRESS-SESSIONS

Ahora vamos a estbleecer la gestión de rutas para la ruta admin y usar las sesiones para ello creamos el archivo admins.js

```
var express = require('express');
var router = express.Router();
module.exports = router;
```

EXPRESS-SESSIONS

Ahora vamos a crear nuestra primera ruta que comprueba si existe la sesión.

```
router.get('/', (req, res, next) => {
    res.status(200).json(req.session.username || "La sesión no se ha creado
});
```

EXPRESS-SESSIONS

Vamos a crear nuestra primera sesión.

```
router.get('/create', (req, res, next) => {
    req.session.username = "xrodriguez";
    res.redirect('/admins');
});
```

EXPRESS-SESSIONS

Como podemos eliminar un elemento de una sesión

```
 router.get('/remove', (req, res, next) => {
    req.session.username = null;
    res.redirect('/admins');
});
```

EXPRESS-SESSIONS

Para eliminar por completo una sesión con todos sus parámetros

```
 router.get('/destroy', (req, res, next) => {
    req.session.destroy();
    res.redirect('/admins');
});
```

GEEKSHUBS Academy_

Aumentado la funcionalidad con flash-sessions

SESIONES FLASH

Son aquellas sesiones que son ejecutadas en la primera redireccion después de ser creadas.

SESIONES FLASH

Instalamos la librería: connect-flash

```
npm install connect-flash --save
```

SESIONES FLASH

Ahora configuramos node para utilizar flash sessions

```
//Cargamos la librería.
var flash = require('connect-flash');
//Cargamos la ruta para las sesiones flash
var login_flash = require('./routes/login_flash');
//Configuramos estas nuevas rutas a Express.
app.use('/login_flash', login_flash);
```

SESIONES FLASH

Ahora le índicamos a Express el uso las sesiones flash

```
//Añadimos la siguiente línea despues de configurar las sesiones express.
app.use(flash());
```

SESSIONES FLASH

Creamos nuestra nueva archivo para enrutar para el tratamiento de sesiones flash

```
var express = require('express');
var router = express.Router();
module.exports = router;
```

SESSIONES FLASH

Creamos nuestra primera ruta para comprobar la sesión flash

```
router.get('/', (req, res, next) => {
    res.send(req.flash('info'));
});
```

SESSIONES FLASH

Creamos una ruta para crear la sesión FLASH

```
router.get('/create', (req, res, next) => {
    req.flash('info', 'Sesión flash info creada');
    res.redirect('/login_flash');
});
```

GEEKSHUBS Academy_

GESTIÓN DE LOGS CON WINSTON

GEEKSHUBS Academy_

Winston

Winston es una librería diseñada para el registro simple y universal con soporte para múltiples transportes. Un transporte es esencialmente un dispositivo de almacenamiento para sus registros. Dicho en palabras sencillas nos permiten almacenar mensajes personalizados de seguimiento (al igual que console.log) en un archivo plano o también desplegarlo por consola.

Winston

Primero instalamos winston mediante consola

```
npm install winston --save
npm install app-root-path --save
```

Winston

Configurando Winston, para ello creamos en nuestro proyecto una nueva carpeta denominada config y crearemos un archivo winston.js y crearemos otra carpeta logs donde se almacenara los logs

Winston

Configuramos nuestro archivo winston.js


```
const appRoot = require('app-root-path');
const winston = require('winston');

var options = {
    file: {
        level: 'info',
        filename: `${appRoot}/logs/app.log`,
        handleExceptions: true,
        format: winston.format.json(),
        maxsize: 5242880,
        maxFiles: 5,
    },
    console: {
        level: 'debug',
        handleExceptions: true,
        format: winston.format.combine(
            winston.format.colorize(),
            winston.format.simple()
        )
    }
};

var logger = winston.createLogger({
    transports: [
      new winston.transports.File(options.file),
      new winston.transports.Console(options.console)
    ],
    exitOnError: false,
});

logger.stream = {
    write: function (message, encoding) {
```

```
        logger.info(message);
    }
}
```

Winston

Los elementos para configurar un archivo son:

- level -> Nivel de los mensajes para el log.
- filename -> La ruta al archivo donde almacenar los logs.
- handleExceptions -> Log de unhandled expcetions.
- maxsize -> Máximo tamaño de almacenamiento del arhcivo
- maxFiles -> Número máximo de archivos a almacenar.
- format -> Formato al que queremos que transforme los logs.

Winston

Los logs se configuran en los siguientes niveles:

- 0: Error
- 1: Warn
- 2: Info
- 3: Verbose
- 4: debug
- 5: silly

si indicamos 'info' únicamente atraparemos los de igual o menor nivel, que en este caso serían: 2-info, 1-warn, 0-error.

Winston

Ahora utilizamos winston dentro de nuestra aplicación

```
var winston = require('./config/winston');
```

Winston

Ahora para poder registrar las peticiones propias del servidor realizamos la siguiente modificación en app.js

```
//Comentamos
//app.use(logger('dev');
//Añadimos la siguiente línea
app.use(logger('combined', { stream: winston.stream }));
```

Winston

Ahora ya podemos utilizar Winston en nuestra Webapp

```
//Cargamos la librería.
var winston = require('../config/winston');
//En el archivo login_flash.js recien creado añadimos al router.get('/',..
winston.info("Local de login-flash");
```

GEEKSHUBS Academy_

NODEMAILER

NodeMailer

NodeMailer es un módulo que nos permite enviar emails, es un proyecto que se inició en el año 2010, y tiene licencia MIT

NodeMailer

Dispone las siguientes características:

- Altamente enfocado en la seguridad, preveniendo vulnerabilidades.
- Puedes usar varios tipos de contenido como lo son Texto plano y HTML.
- Te permite adjuntar archivos con el envío de los mensajes.
- Diferentes métodos de transporte como lo son MANDRILL, SENGRID, etc..

NodeMailer

Requiere una versión superior a la 6 de node.js

Disponemos de una amplia cantidad información en la página del proyecto

NodeMailer

Vamos a instalar el módulo:

```
$ npm i -S nodemailer
```

NodeMailer

Creamos una carpeta config, y generamos un archivo de configuración de nodemailer este archivo lo llamaremos emailConf.js

NodeMailer

Ahora configuramos nodeMailer

```
const NodeMailer= require('nodemailer');
let email={};
module.exports = email;
```

NodeMailer

Ahora estableceremos las pasarela de transporte vamos a configurar un servicio con gmail

```
email.transporter = NodeMailer.createTransport({
    service:'Gmail',
    auth: {
        user: '',
        pass: ''
    },
  },
  {
    from:'jaroso@gmail.com',
    headers: {
    }
})
```

NodeMailer

Ahora vamos a utilizar nodemailer dentro de nuestro archivo de rutas

```
const Email = require('../config/email');
let message= {
    to:'xavi@geekshubs.com',
    subject : 'Email de prueba',
    html: 'Hola es una prueba'
  }
  Email.transporter.sendMail(message,(error,info) =>{
    if(error){
        res.status(500).send(error, message);
        return
      }
    Email.transporter.close();
    res.status(200).send('Respuesta "%s"' + info.response);
    })
```

NodeMailer

Como podemos añadir imágenes o archivos en nuestro emails.

Para ello utilizaremos la proiedad attachments

```
let message= {
  to:'xavi@geekshubs.com',
  subject : 'Email de prueba',
  html: 'Hola es una prueba',
  attachments:[
    {
      filename:'super.jpeg',
      path:__dirname +'/../public/images/super.jpeg',
      cid: 'imagen'
    }
  ]
}
Email.transporter.sendMail(message,(error,info) =>{
  if(error){
      res.status(500).send(error, message);
      return
    }
  Email.transporter.close();
```

```
    res.status(200).send('Respuesta "%s"' + info.response);
    })
```

NodeMailer

Simplemente para adjuntar debemos hacer lo siguiente.

```
{
        filename:'super.jpeg',
        path:__dirname +'/../public/images/super.jpeg'
}
```

NodeMailer

Como enviar emails formateados utilizando handlebars

Para ello vamos a utilizar un módulo denominado nodemailer-express-handlebars

```
$ npm i -S nodemailer-express-handlebars
```

NodeMailer

Ahora realizamos los cambios necesarios para utilizar este nuevo sistema de plantillas

```
const Hbs = require('nodemailer-express-handlebars');
const Path = require('path'); //Módulo para registrar el path
```

NodeMailer

Ahora realizamos las modificaciones necesarias en el transporter, pero crearemos un nuevo directorio denominado dentro de vistas email-templates y generaremos una nueva template denominado email.hbs

NodeMailer

Ahora añadimos el contexto de nuestra plantilla la tratarse de handlebars podemos utilizar toda la potencia de este motor de plantillas.

```
{{ texto }}Mas prueba
```

NodeMailer

Realizamos modificaciones a nuestro transporte para que utilize hbs


```
Email.transporter.use('compile', Hbs ({
  viewEngine: 'hbs',
  extName:'.hbs',
  viewPath: Path.join(__dirname,'../views/email-templates')
}));
let message= {
  to:'xavi@geekshubs.com',
  subject : 'Email de prueba',
  template:'email',
  context: {
    text:'Enviamos una prueba por handlebars'
  },
  attachments:[
    {
      filename:'super.jpeg',
      path:__dirname +'/../public/images/super.jpeg',
      cid: 'imagen'
    },
    {
      filename:'super.jpeg',
```

```
      path:__dirname +'/../public/images/super.jpeg'
    }
  ]
```

GEEKSHUBS Academy_

MULTER

MULTER

Multer es un módulo que nos permite subir archivos a node.js con express de forma sencilla

Tenemos toda la información en la página del proyecto

MULTER

Ahora instalamos las dependencias necesarias

```
$ npm i -S multer
```

MULTER

Creamos un directorio donde almacenaremos los archivos que vamos a subir por ejemplo uploads

MULTER

Configuramos como Multer debe almacenar los archivos

```
const Multer = require('multer');
const storage = Multer.diskStorage({
  destination: (req, file, cb)=>{
    cb(null,"uploads/");
  },
  filename: (req, file, cb) =>{
    cb(null, file.originalname);
  }
});
const upload = Multer({storage});
```

GEEKSHUBS Academy_

MULTER

Ahora configuramos nuestro formulario para subir los archivos

MULTER

Ahora simplemente generamos la ruta y utilizamos el middleware para la subida de los archivos

```
router.post('/upload',upload.single('file'),(req, res, next)=>{
  res.json(req.file);
})
```

GEEKSHUBS Academy_

VARIABLES DE ENTORNO

VARIABLES DE ENTORNO

Las variables de entorno nos permite definir parámetros básicos en la configuración de nuestros proyectos. Así como diferentes entornos:

- Desarrollo.
- Producción.

VARIABLES DE ENTORNO

Podemos enviar entorno mediante el objeto **process.env** por ejemplo para estableer el puerto de escucha de nuestro servidor node.js

```
//Linux or Mac
$ PUERTO=3001 node app.js
//Windows
c\> SET PUERTO =3001
c\> node app.js
```

VARIABLES DE ENTORNO

En el ejemplo anterior enviamos una variables de entorno denoinado puerto, para que pueda ser recogida por nuestra aplicación.

```
let puerto = proess.env.PUERTO || 3000;
```

Nos generamos un nuevo puerto y en caso de no pasarle la variables nos lo genera en el puerto 3000

VARIABLES DE ENTORNO

En este momento podemos gestionar las variables de entorno generando un archivo que en función nos ejecute una configuración u otra por ejemplo:

```
let config= {
    "desarrollo":{
        "SERVER":"http://192.168.34",
        "PORT": 3000
    },
    "produccion":{
        "SERVER": "http://app.node.es",
        "PORT": 80
    }
};

module.exports= config;
```

VARIABLES DE ENTORNO

Ahora ya podemos cargar una configuración u otra en función de la variable de entorno recibida

```
let env=process.env.NODE_ENV|| 'desarrollo';
let config = require('./config')[env];
switch(env){
  case 'desarrollo':
    console.log(config.SERVER);
    console.log(config.PORT);
    break;
  case 'produccion':
    console.log(config.SERVER);
    console.log(config.PORT);
    break;
}
```

VARIABLES DE ENTORNO

Ahora sólo debemos cargar la aplicación ínicando la variable de entorno **NODE_ENV** adecuada

```
$ NODE_ENV=produccion npm start
```

GEEKSHUBS Academy_

SISTEMA DE PAGINACIÓN.

SISTEMA DE PAGINACIÓN

Vamos a ver como gestionar un tabla con paginación en el caso de datos extensos.

Para ello primero debemos generar una consulta dentro de nuestro modelo que nos devuelva un numéro de filas de la consulta y el número total de filas para poder saber en que situación nos encontramos.

SISTEMA DE PAGINACIÓN

```
Films.paginate =(offset, limit, cb)=>{
  if(!conn) return cb("No se ha podido crear conexión");
  conn.query('SELECT * FROM  film LIMITS ?, ?', [offset, limit],(error,row
    if(error){
      return cb(error);
    }
    conn.query('SELECT COUNT(*) as total FROM users',(error, count)=>{
      return callback(null,{count,rows});
    })
  })
}
```

SISTEMA DE PAGINACIÓN

Para poder paginar debemos utilizar helpers en Handlebars

Los Helpers son una manera que dispone handlebars para ampliar su funcionamiento añadiendo nuevas propiedades al mismo.

Para ello vamos a crear un nuevo directorio llamado **Helpers** y dentro del mismo un archivo denominado hbs.js

SISTEMA DE PAGINACIÓN

Primero debemos registrar la nueva librería para poder utilizarlo posteriormente en las vistas.

```
module.exports=(hbs)=>{

}
```

SISTEMA DE PAGINACIÓN

Para registrar el helper utilizamos el método **registerHelper**

Nuestro primer helper es para comprobar si la página es activa.

```
hbs.registerHelper('is_active', (index, number) => {
        return (index + 1) == number ? 'active' : '';
    });
```

SISTEMA DE PAGINACIÓN

Los siguientes helpers nos permite comprobar la página anterior y posterior

```
hbs.registerHelper('prev_link', (pagination) => {
    return pagination.href(true);
});
hbs.registerHelper('next_link', (pagination) => {
    return pagination.href();
});
```

SISTEMA DE PAGINACIÓN

El siguiente Helper nos ayudará a comprobar si existen más páginas.

```
hbs.registerHelper('has_next_links', (pageCount, pagination, options) =>
      if(pagination.hasNextPages(pageCount)) {
          return options.fn(this);
      }
      else
      {
          return options.inverse(this);
      }
})
```

SISTEMA DE PAGINACIÓN

Ahora vamos a crear la lógica necesaria para desarrollar la paginación suficiente para ello primero

debemos instalar un módulo que se denomina **express-paginate**

```
$ npm i -S express-paginate
```

Este módulo es un midleware que nos permite gestionar la paginación.

SISTEMA DE PAGINACIÓN

Ahora vamos a utilizar el módulo dentro de nuestro proyecto, así como cargar el helper que hemos creado.

```
var paginate = require('express-paginate');
app.use(paginate.middleware(2,20));
//Después de la definición de hbs añadimos los helpers.
require('./helpers/hbs')(hbs);
```

SISTEMA DE PAGINACIÓN

Creamos una nueva ruta para gestionar esta paginación.

En la que creamos los diferentes elementos para gestionar la paginación.

```
let page=(parseInt(req.query.page) || 1) -1; //Obentemos la situación de l
let limit = 20; // Límite de  elementos por página.
let offset = page * limit ;  // El elemento de cáculo de páginas
```

SISTEMA DE PAGINACIÓN

En cada petición podemos gestionar y recalcular el número y posición.

```
Films.paginate(offset, limit, (error, films)=>{
    if(error){
        return res.status(500).send(error);
    }
    const currentPage = offset ===0 ? 1:(offset/limit)+1;
    const totalCount = films.count[0].total;
    const pageCount = Math.ceil(totalCount /limit);
    const pagination = paginate.getArrayPages(req)(10,pageCount, currentPa
    res.render('pagination',{
        films: films.rows,
        currentPage,
        links: pagination,
        hasNext: paginate.hasNextPages(pageCount),
        pageCount
    });
})
```

SISTEMA DE PAGINACIÓN

Ahora vamos a ver y gestionar la vista de la página.

En la priemra sección mostramos el listado de páginas.

```
{#each films }}

            {{ film_id }} - {{ title }}

            Title: {{ title }}, Description: {{ description }}

            Release Year: {{ release_year }}

{{/each}}
```

SISTEMA DE PAGINACIÓN

Ahora gestionamos la paginación.

```
{{#with paginate}}
    {{#if hasPreviousPages}}

            Anterior

    {{/if}}
{{/with}}
```

SISTEMA DE PAGINACIÓN.

Ahora obtenemos el número de paǵinas.

```
{{#each links}}

        {{number}}

{{/each}}
```

SISTEMA DE PAGINACIÓN.

Y así obtenemos la siguiente enlace

```
{{#with paginate}}
    {{#has_next_links ../pageCount this}}

            Siguiente

    {{/has_next_links}}
{{/with}}
```

GEEKSHUBS Academy_

MULTIIDIOMA

MULTIIDIOMA

Uno de los elementos importantes es poder tener multiidioma implantado en nuestra web, para ello usaremos el módulo i18n

```
$ npm i -S i18n
```

MULTIIDIOMA

Ahora configuramos y añadimos la ruta y el módulo adecuado para funcionar.

```
//Cargando la librería.
var i18=require('i18n');
//Cargando las rutas.
var i18n_routes = require('./routes/i18n-routes');
```

MULTIIDIOMA

Ahora configuramos nuestro módulo en APP.JS

```
i18n.configure({
  locales:['es','en'],
  cookie:'secret-lang',
  directory: __dirname+'/locales',
  defaultLocale:'es'
})
```

MULTIIDIOMA

Ahora creamos un directorio denominado locales donde crearemos dos json con los idiomas definidos.

- en.json
- es.json

En dichos archivos definiremos los distintos elementos para nuestras páginas.

MULTIIDIOMA

```
//en.js
{
  "hello": "hello",
  "hello %s": "hello %s",
  "%d product": {
    "one": "%d product",
    "other": "%d products"
  }
}
```

MULTIIDIOMA

```
//es.js
{
  "hello": "hola",
  "hello %s": "hola %s",
  "%d product": {
    "one": "%d producto",
    "other": "%d productos"
  }
}
```

MULTIIDIOMA

Ahora vamos a crear una ruta para modificar el idioma de nuestra aplicación.

```
app.get('/locale/:lang',(req,res,next)=>{
  res.cookie(
    'secret-lang',
    req.params.lang
  );
  res.redirect('i18n-routes');
});
app.use(i18n.init);
```

MULTIIDIOMA

Ahora creamos la ruta.

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

router.get('/', (req, res, next) => {
    res.render('lenguaje', {
        title: 'i18n'
    });
});

module.exports = router;
```

Ahora creamos la vista para presentar en pantalla

```
{{__ "hello %s" "Xavi Rodriguez" }}
{{{__n "%d product" "%d products" 20}}}
```