

# APLICACIONES DE INTERNET

## PRÁCTICA 2

Martínez Lona Verónica Montserrat

Para la creación del web service se utilizó NodeJs, MongoDB y Express.

Los pasos más relevantes del desarrollo del servicio son:

1. Creación del proyecto con express
2. Creación de las vistas
3. Creación del esquema de la base de datos
4. Creación de la API
5. Consumo de la API

### 1. Creación del proyecto con express

La estructura del proyecto es la siguiente:

- **Controlers** //Controlador de los empleados
- **Models** //El modelo a utilizar en la base de datos
- **Public**
  - Images //Imágenes pertenecientes al diseño de las vistas
  - Javascripts // Consumo de la API
  - Stylesheets // Hojas de estilo para el diseño de las vistas
- **Routes** //Rutas a cargar
- **Vistas** //Archivos hbs para el diseño de la interfaz
- **App.js** //Servidor

En las rutas tenemos los procedimientos que se pueden realizar en cada una de ellas:

- **Index** //POST(No implementado)
- **Employee** // GET, PUT, DELETE
- **Employee\_add** //POST

#### App.js

Servidor de la aplicación, no es muy distinto de lo que hemos trabajado en clases., por ello no comentaré todas las líneas de código.

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var exphbs = require('express-handlebars');
var methodOverride = require('method-override');
var mongoose = require('mongoose');
var employeeManager = require('./controllers/employeeManager.js');
```

Importamos las dependencias que utilizaremos en el proyecto. Yo utilice express generator para comenzar el poyecto.

```
//Endpoints
var login = require('./routes/index');
var employee = require('./routes/employee');
var add = require('./routes/add_employee');
```

Estos son los end-points que se utilizaron en todo el proyecto.

---

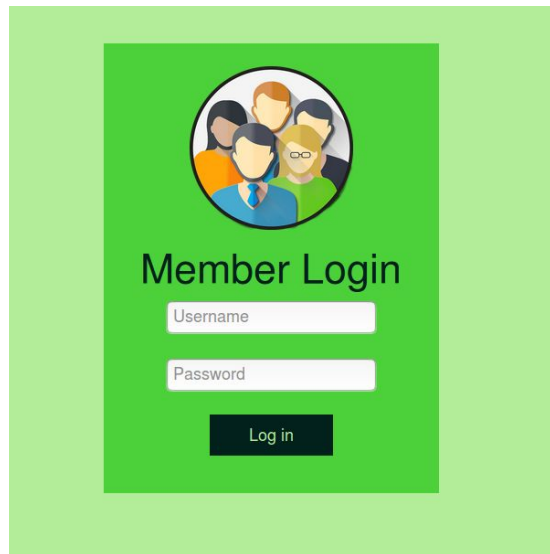
## 2. Creación de las vistas

Las vistas son:

- Login.hbs
- Employee.hbs
- Edit\_employee.hbs
- Add\_employee.hbs

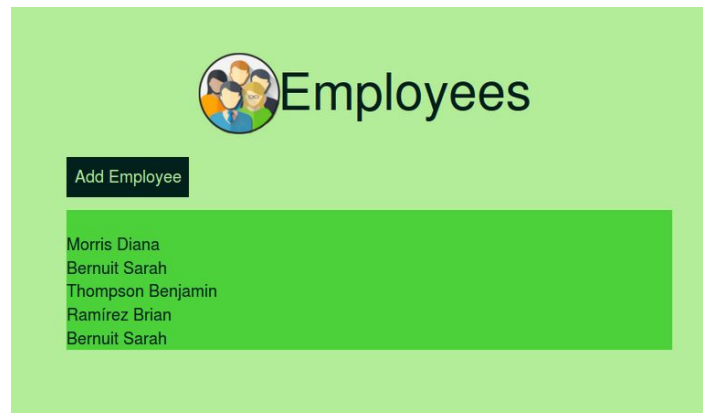
Las vistas están hechas con handlebars + css. El diseño es simple, al igual que la funionalidad.

### Login.hbs



No está implementado el login, sin embargo al pulsar el boton *Log in* redirecciona a la ruta */employee*.

## Employee.hbs



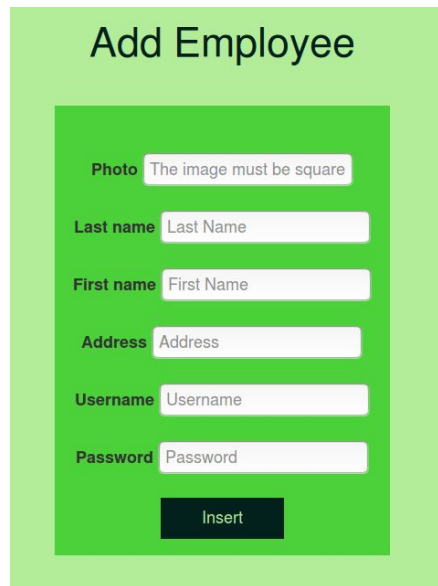
Aquí se muestra una lista de los empleados en la base de datos. Existe un botón de agregar que redirecciona a la página de agregar empleado. Al clicar sobre un empleado debería redireccionarse a su página de empleado específica (esto no funciona correctamente).

## edit\_employee.hbs

The screenshot shows a web page with a light green background. At the top center, there is a dark green button with the text "Edit Employee" in white. Below this, on the left, is a white rectangular box containing a photograph of a woman with long blonde hair. To the right of the photo is a form with several input fields: "Last name" with the value "Bernuit", "First name" with the value "Sarah", "Address" with the value "3290", "Username" with the value "SarahB", and "Password" with the value "...". At the bottom of the form are two dark green buttons: "Update" and "Delete".

En esta vista se carga la información de un usuario específico, todos los datos pueden ser actualizados, salvo la fotografía. En esta misma vista se elimina el empleado.

add\_employee.hbs

A screenshot of a web form titled "Add Employee" with a light green background. The form itself has a darker green background and contains several input fields: "Photo" with a placeholder "The image must be square", "Last name" with "Last Name", "First name" with "First Name", "Address" with "Address", "Username" with "Username", and "Password" with "Password". At the bottom of the form is a black button labeled "Insert".

Este es el formulario para agregar un empleado. No es necesario que se llenen todos los campos.

### 3. Creación del esquema de la base de datos

```
var employeeSchema = new Schema({
  photo: String,
  lastName: String,
  firstName: String,
  address: String,
  username: String,
  password: String
});
```

El esquema es realmente simple.

## 4. Creación de la API

controllers/employeeManager.js

```
var employee = require('../models/employee.js');
```

Puesto que se va a manejar los documentos de la base de datos, es necesario importar el modelo de *employee*.

```
//GET
exports.findAll = function(req, res) {
  employee.find(function(error, employeeList) {
    if(error) { return res.send(500, error.message); }
    return res.render('employee', {
      employee: employeeList
    });
    return res.status(200).jsonp(employeeList);
  });
};
```

La primera función que se implementó fue *findAll(GET)*, que permite ver todos los empleados. Lo importante de aquí es que se hace el render de la vista *'employee'*, y enviando *employeeList* para mostrarlo al usuario.

```
//GET
exports.findById = function(req, res) {
  var id = req.params.id;
  console.log('Employee: ' + id);
  employee.findById(req.params.id, function(error, found) {
    if(error) { return res.send(500, console.error.message); }
    return res.render('edit_employee', {
      employee: found
    });
    return res.status(200).json(found);
  });
};
```

La segunda función fue *findById(GET)*, que permite ver los datos específicos de un empleado. El request viene con los datos pertenecientes al empleado seleccionado, lo que necesitamos para filtrar es el id, por ello lo obtenemos de *req.params.id*. Para obtener los datos de ese único id es necesario pasarlo como parámetro de la función *employee.findById*. Finalmente se rendera la vista *'edit\_employee'*, con los campos de un formulario rellenos con los datos del empleado seleccionado.

```
//PUT
exports.updateEmployee = function(req, res) {
  var id = req.params.employee_id;
  employee.findById(id, function(error, selectedEmployee) {
    if(error) { return res.send(500, console.error.message); }
    //updateinfo
    selectedEmployee.photo = req.body.photo;
    selectedEmployee.lastName = req.body.lastName;
    selectedEmployee.firstName = req.body.firstName;
    selectedEmployee.address = req.body.address;
    selectedEmployee.username = req.body.username;
    selectedEmployee.password = req.body.password;
    //save the employee
    selectedEmployee.save(function(error) {
      if(error) { return res.send(500, console.error.message); }
      employee.find(function(error, employeeList) {
        if(error) { return res.send(500, error.message); }
        return res.render('employee', {
          employee: employeeList
        });
      });
      return res.status(200).jsonp(employeeList);
    });
  });
});
};
```

La tercera función es *updateEmployee* (PUT). Similar al anterior (de hecho, se utiliza aquí). Después de que se llena la información de un usuario específico, al presionar el botón de Update se prosigue a tomar los datos del formulario y actualizar con ellos la información de la base de datos. Para conseguir esos datos se hace por medio de *selectedEmployee.campo*. Finalmente guardamos los valores y rendiremos la vista de empleados, ya actualizada.

```
//POST
exports.addEmployee = function(req, res) {
  console.log(req.body);

  //Create a new instance of the employee model
  var newEmployee = new employee({
    employeeId: req.body.employeeId,
    photo: req.body.photo,
    lastName: req.body.lastName,
    firstName: req.body.firstName,
    address: req.body.address,
    username: req.body.username,
    password: req.body.password
  });

  //Save the employee and check for errors
  newEmployee.save(function (error) {
    if(error) { return res.send(500, error.message); }
    employee.find(function(error, employeeList) {
      if(error) { return res.send(500, error.message); }
      return res.render('employee', {
        employee: employeeList
      });
    });
    return res.status(200).jsonp(employeeList);
  });
});
};
```

La siguiente función es *addEmployee*, que, como su nombre dice, sirve para añadir un empleado a la base de datos. Se abre un formulario dónde se rellenan libremente los campos del fomulario, que después de obtenerse del cuerpo del request. Se guardan en la base de datos y finalmente se renderea la vista de principal de *employee*.

```
exports.deleteEmployee = function(req, res) {  
  employee.remove({ _id: req.params.employee_id }, function(error, selectedEmployee) {  
    if(error) { return res.send(500, error.message); }  
    employee.find(function(error, employeeList) {  
      if(error) { return res.send(500, error.message); }  
      return res.render('employee', {  
        employee: employeeList  
      });  
      return res.status(200).jsonp(employeeList);  
    });  
  });  
};
```

Por último se definió la función *deleteEmployee*, que nos permite eliminar un empleado específico. Para poder obtenerlo específicamente sólo es necesario guardar y enviar el valor del id del empleado seleccionado. Después de que fue añadido es renderear la vista de principal de empleado.

Es uno de los archivos más importantes del proyecto, puesto que es la **RestFullAPI** que utilizaremos.

---

## 5. Consumo de la API

Cada procedimiento se llama en un js correspondiente a la ruta que maneja la vista.

Se tienen 3 rutas:

- Add\_employee
- Employee
- index

En cada uno de estos archivos (salvo *index*) se tuvo que importar el *employeeManager*, pues es la API, que nos permite el manejo de la base de datos.

**INDEX**

```

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

/* GET login page. */
router.get('/', function(req, res, next) {
  return res.render('login');
});

module.exports = router;

```

La ruta index no tiene otro procedimiento más que *GET*, que sólo rendera la vista de *login*.

## EMPLOYEE.JS

```

var express = require('express');
var router = express.Router();
var employeeManager = require('../controllers/employeeManager.js');

/* GET employee page. */

router.get('/', employeeManager.findAll); //READ

router.get('/:id', employeeManager.findById);
router.put('/:id', employeeManager.updateEmployee);
router.delete('/:id', employeeManager.deleteEmployee);

module.exports = router;

```

Es la ruta con más procedimientos.

*router.get('/', employeeManager.findAll);*

Carga en la vista principal de employee una lista de los empleados.

*router.get('/:id', employeeManager.findById);*

Después de seleccionar un empleado específico (*/:id* ←se refiere al id del empleado seleccionado) carga la vista de *edit\_employee* con la información del empleado seleccionado.

*router.put('/:id', employeeManager.updateEmployee);*

Después de cargar la información y al apretar el botón *Update* se actualizan los datos del empleado seleccionado.

*router.delete('/:id', employeeManager.deleteEmployee);*

Llama al procedimiento encargado de eliminar al usuario específico.



## ADD\_EMPLOYEE.JS

```
var express = require('express');
var router = express.Router();
var employeeManager = require('../controllers/employeeManager.js');

/* GET employee page. */
router.get('/', function(req, res, next) {
  res.render('add_employee');
});

/*POST new employee*/
router.post('/', employeeManager.addEmployee);

module.exports = router;
```

El método get de esta parte no hace más que cargar la vista del formulario para insertar un nuevo empleado. Mientras que el método post es para agregar un nuevo empleado.