Universidad de La Laguna

Práctica de laboratorio de arquitectura del software.

Desarrollo de Sistemas Informáticos- Grado en Ingeniería Informática

- **Darwin González Suárez**
- **Cristina Pacheco González**
- **Javier Castro González**
- **Daniel Alberto Melián León**

Enlace prototipo práctica: https://github.com/DarwinGonzalez/PracticaArqSoftDSI

Tipo de aplicación

El tipo de aplicación elegido para el proyecto se basa en una aplicación en la que el usuario pueda introducir su película favorita (con una serie de campos relacionados con la película) y esta será guardada en una base de datos y mostrada en una tabla en la plataforma para que cualquier otra persona pueda visualizar estas películas, y en base a las puntuaciones ver o no la película si le apetece.

Infraestructura y estrategia de despliegue

La **infraestructura** utilizada está formada por un servidor, donde tendríamos guardada tanto la base de datos como el código fuente del proyecto, siendo desplegada en la dirección IP de este servidor. Para las pruebas que se han realizado se ha utilizado el un portátil como servidor local que aloja tanto el código como el servidor de MongoDB donde se guardan los datos relacionados con las películas.

Para el **despliegue** del proyecto es necesario ejecutar los siguientes comandos en la terminal. En primer lugar es necesario descargar el repositorio y posicionarnos dentro de la carpeta. Una vez ahí ejecutamos el siguiente comando para instalar los diferentes módulos y dependencias necesarios(explicados más adelante):

```
npm i
```

Una vez realizado esto, de cara a desplegar el prototipo, ejecutaremos el siguiente comando para poder acceder a él a través del navegador:

```
npm run start
```

Finalmente nuestro proyecto estará disponible en: http://localhost:3000/pelicula

Las tecnologías usadas

La herramienta que utilizamos como base para empezar a trabajar y facilitar el trabajo en grupo fue un sistemas de control de versiones, en este caso GitHub.

Por otro lado, para el desarrollo de todo el código del proyecto utilizamos diferentes lenguajes. Las tecnologías utilizadas las dividimos en dos grupos:

- **FRONT - END**

Las tecnologías empleadas en la parte del Front-End y que por consiguiente corren del lado del cliente han sido:

En el caso de HTML, se ha utilizado para definir la estructura de la página web.

Una vez que tengamos las estructura definida con el lenguaje HTML, pasamos a la parte de diseño, utilizando para ello el lenguaje CSS, añadiendo estilos para darle una mejor apariencia a la página.

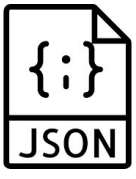
Por último, incorporamos JavaScript con el objetivo de crear diferentes efectos atractivos y dinámicos en la página, interactuando así con los usuarios.

Para aplicar el uso de las 3 anteriores tecnologías explicadas, se ha utilizado una plantilla de Bootstrap para la parte de las vistas o página principal donde se muestra el contenido de la plataforma web.

- **BACK - END**

En la parte del back-end tenemos las tecnologías que corren en el lado del servidor e interactúan con la base de datos del proyecto.

Para ello, se ha utilizado NodeJs que es un lenguaje basado en JavaScript pero que que funciona solo en el lado del servidor y con el que se ha trabajado toda la parte del servidor, estableciendo la comunicación con la base de datos.



Por otro lado, la base datos usada ha sido MongoDB. Es una base de datos NoSQL orientadas a documentos(JSON), la cual la hemos elegido ya que nuestro proyecto no necesitamos el uso de transacciones, ni tampoco consultar los datos relacionados entre dos o más colecciones.

Atributos básicos de la aplicación

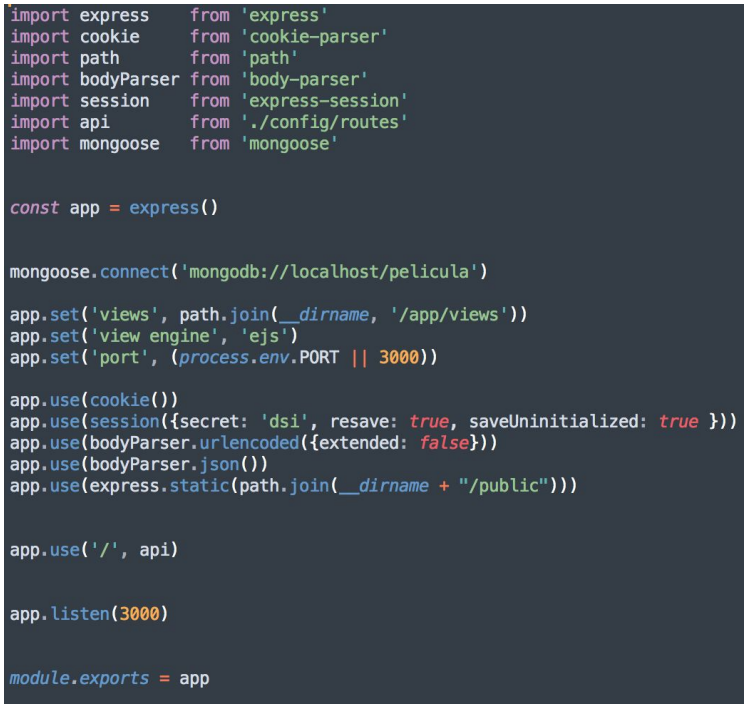
En cuanto a los diferentes atributos con los que cuenta nuestra aplicación hay varios:

- **SERVIDOR**

En primer lugar tenemos la parte del servidor, que es la encargada de atender el conjunto de peticiones que se hacen al servicio, y como no, de levantar el servicio en el puerto indicado. Además de esto, el servidor es el encargado de cargar las vistas (elemento que ahora nombraremos con más detalle).

En la imagen de abajo vemos como tenemos configurado el servidor. Lo primero de todo, es importar las librerías o módulos que hemos instalado y que nos permitan trabajar de una manera más cómoda con la comunicación cliente-servidor. Utilizamos **Express** que es un mini framework de **NodeJs** que nos proporciona un conjunto de características para las aplicaciones web y nos permite crearlas de manera sencilla. Luego para qué **Express** entienda las peticiones con un **JSON** sin ningún problema, utilizamos el módulo **body-parser**. Por otra parte, como ya dijimos anteriormente, la base de datos que empleamos es **MongoDB**, y para definir el esquema que va a tener nuestra base de datos y guardarlo en ella, hacemos uso de **mongoose**. Para el tratamientos de las cookies, utilizamos **cookie-parser** que nos facilita su creación y consulta en nuestra página. Para el uso de variables de sesión, en la que necesitamos para almacenar datos que no se borren entre una y otra

página que solicita el cliente, utilizamos el módulo **express-session**. También, importamos **path** que lo utilizamos para hacer operaciones con strings que contienen URLs y rutas de archivos. Y por último, importamos **api**, la cual será la variable que contiene las diferentes rutas asociadas a una serie de funciones implementadas en los controladores. A continuación podemos ver una imagen con el contenido del index.js o servidor principal.



```
import express    from 'express'
import cookie     from 'cookie-parser'
import path       from 'path'
import bodyParser from 'body-parser'
import session    from 'express-session'
import api        from './config/routes'
import mongoose   from 'mongoose'


const app = express()


mongoose.connect('mongodb://localhost/pelicula')

app.set('views', path.join(__dirname, '/app/views'))
app.set('view engine', 'ejs')
app.set('port', (process.env.PORT || 3000))

app.use(cookie())
app.use(session({secret: 'dsi', resave: true, saveUninitialized: true }))
app.use(bodyParser.urlencoded({extended: false}))
app.use(bodyParser.json())
app.use(express.static(path.join(__dirname + "/public")))


app.use('/', api)


app.listen(3000)


module.exports = app
```

index.js

Todos los módulos nombrados(excepto path y api), están reflejados en el archivo ***package.json***, todos ellos son necesarios para que la aplicación funcione.

```
{
  "name": "dsi",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "babel-node -- index",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.2",
    "cookie-parser": "^1.4.3",
    "ejs": "^2.5.8",
    "express": "^4.16.3",
    "express-session": "^1.15.6",
    "mongoose": "^5.0.14"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-core": "^6.26.0",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1"
  }
}
```

package.json

Por otra parte, vemos que en las dependencias del fichero ***package.json*** aparece el motor de plantillas *EJS*, basado en **JavaScript** y que nos permite generar código HTML. En el fichero ***index.js*** también aparece reflejado, indicando le al servidor que vamos hacer uso de este motor de plantillas:

```
app.set('view engine', 'ejs')
```

Y también le indicamos donde se encuentra los archivos de la plantilla, en este caso, en el directorio **/app/views:**

```
app.set('views', path.join(__dirname, '/app/views'))
```

Además de todo lo anterior, es importante indicarle al servidor donde queremos arrancar o iniciar la aplicación. Para ello, arrancamos la aplicación con ***app.listen***, y le decimos porque puerto queremos que corra el servidor (**3000**).

- ## Directorio /app

Por otra parte tenemos nuestra aplicación en el directorio ***/app*** y dentro de la misma otros tres subdirectorios:

```
├── app
│   ├── controllers
│   │   └── peliculaController.js
│   ├── models
│   │   └── pelicula.js
│   └── views
│       └── pelicula.ejs
```

Models

En este directorio, se define el modelo o esquema que va a tener los datos que se van almacenar nuestra base de datos y sobre la cual se va interactuar.

```
import mongoose from 'mongoose'
const Schema = mongoose.Schema

const Pelicula = new Schema({
    nombre:     { type: String },
    director:   { type: String },
    genero:     { type: String },
    puntuacion: { type: Number},
    estreno:    { type: String},
    duracion:   {type: Number}

});


module.exports = mongoose.model('Pelicula', Pelicula);
```

Views

En este sitio se encuentran las plantillas de las páginas y serán usadas por el motor de plantillas elegido, que en nuestro caso ha sido ***EJS***. Se encargará de cargar el contenido multimedia que ve el usuario.

```
    <h2> Peliculas: </h2>
    <div class="container">
          <table class="table table-striped">
            <thead>
              <tr>
                <th>Nombre</th>
                <th>Director</th>
                <th>Genero</th>
                <th>Puntuacion</th>
              </tr>
            </thead>
            <tbody>
              <%for (var i=0; i<peliculas.length; i++) {%>
              <tr>
                <td><%=peliculas[i].nombre%></td>
                <td><%=peliculas[i].director%></td>
                <td><%=peliculas[i].genero%></td>
                <td><%=peliculas[i].puntuacion%></td>
              </tr>
               <% } %>
            </tbody>
          </table>
</div>
```

Controllers

En el intermediario entre la vista y el modelo, encargado de determinar las funciones que se podrán hacer.

```
import Pelicula        from '../models/pelicula.js'


async function main(req, res) {
    let peliculas = await Pelicula.find({})
    res.render('pelicula', { peliculas: peliculas })
}

async function getPelicula(req, res) {
    const pelicula = new Pelicula({ 'nombre': req.body.nombre, 'director': req.body.director, 'genero': req.body.genero,
    let result = await Pelicula.findOne({'nombre': req.body.nombre})
    if (result) throw new Error('Pelicula ya existe')
    pelicula.save()

    res.redirect('/pelicula')
}

module.exports = { main, getPelicula }
```

- **Directorio /config**

En este directorio tenemos el fichero *routes.js* en el que se describen todas las posibles rutas accesibles de la web y las acciones que se determinan para cada una de ellas.

```
import peliculaController from '../app/controllers/peliculaController'
import express        from 'express'

const api = express.Router()

api.get('/pelicula', peliculaController.main)
api.post('/pelicula', peliculaController.getPelicula)


module.exports = api
```

Definición del estilo arquitectónico

En base a lo comentado en todo lo anterior podemos determinar que nuestra aplicación se trata de una aplicación con una arquitectura de **Modelo Vista Controlador,** también conocida comúnmente como **MVC**. Esto se puede determinar debido a que en una fase previa al diseño se ha hecho una distribución de directorios orientada a cumplir con este modelo de arquitectura tan conocido. En el tenemos diferenciadas estas tres partes cada una con su función determinada.

```
.
├── README.md
├── app
│   ├── controllers
│   │   └── peliculaController.js
│   ├── models
│   │   └── pelicula.js
│   └── views
│       └── pelicula.ejs
├── config
│   └── routes.js
├── index.js
├── npm-debug.log
├── package-lock.json
├── package.json
└── public
```