

**INSTITUTO DE EDUCACIÓN SECUNDARIA CAMPANILLAS**

TÉCNICO SUPERIOR DE DESARROLLO DE APLICACIONES WEB

**PROYECTO FINAL**

**Gamegram**

**FINAL PROJECT**

**Gamegram**

Realizado por

**Alberto López Martín**

Tutorizado por

**Equipo Educativo 2DAW**

**DEPARTAMENTO DE FAMILIA PROFESIONAL DE INFORMÁTICA Y COMUNICACIONES**



MÁLAGA, JUNIO DE 2021

**INDICE**

1. INTRODUCCIÓN
   1. Presentación y objetivos
2. ESPECIFICACIÓN DE REQUISITOS
   1. Funciones del product
   2. Características del usuario
   3. Restricciones
   4. Supuestos y dependencias
   5. Requisitos específicos
3. TECNOLOGÍAS UTILIZADAS EN EL DESARROLLO DEL PROYECTO
4. DOCUMENTACIÓN

# 1. Introducción

## 1.1 Presentación y objetivos

Este documento describe el trabajo realizado en el proyecto final de grado de desarrollo de aplicaciónes web. El proyecto consiste en el desarrollo del sitio web de una página web social. El objetivo de esta web es dar una plataforma donde los usuarios pueden subir imágenes capturadas de sus juegos favoritos.

La web es accesible desde cualquier navegador por Internet y tiene información referente a los juegos y sus miembros.

En cuanto a los usuarios, la página tiene varios tipos. Por un lado, están los usuarios no registrados, que únicamente pueden ver las fotos ya subidas. Por otro, están los usuarios registrados, que pueden acceder a una mayor funcionalidad dentro de la aplicación, pueden editar su perfil de usuario y pueden, subir, editar y eliminar fotos.

# 2.Especificación de Requisitos

La aplicación desarrollada pretende dar una plataforma donde los usuarios pueden subir imágenes capturadas de sus juegos favoritos. Se podrá acceder a la aplicación desde cualquier sistema operativo que tenga conexión a Internet utilizando un navegador web.

## 2.1 Funciones del producto

A continuación se muestran las funciones que conforman la aplicación, según el tipo de usuario que se encuentre conectado:

* Usuario anónimo
  + Autenticación: Se muestra un par de cajas de texto en las que el usuario ha de introducir su nombre de usuario y contraseña para acceder a la parte privada de la aplicación.
  + Registro: Se muestra un par de cajas de texto en las que el usuario ha de introducir su nombre de usuario, email y contraseña para registrar su usuario.
  + Ver fotos: El usuario puede ver las fotos creadas por los usuarios registrados.
  + Buscador: El usuario puede buscar fotos ya creadas.
* Usuario registrado
  + Cerrar sesión: Se muestra un botón para que el usuario cierre su sesión. Cuando se pulsa se le redirige a la página de inicio.
  + Perfil: Se muestra un botón para que el usuario pueda ver su perfil y editarlo una vez dentro.
  + Ver fotos: El usuario puede ver las fotos creadas por los usuarios registrados.
  + Buscador: El usuario puede buscar fotos ya creadas.
  + Crear fotos: El usuario puede subir fotos a través de un formulario.
  + Editar fotos: El usuario puede editar fotos a través de un formulario.
  + Eliminar fotos: El usuario puede eliminar fotos a través de un botón.

## 2.2 Características del usuario

Se puede diferenciar entre dos tipos de usuarios, los usuarios no registrados y los usuarios

registrados.

* Usuarios no registrados: Este tipo de usuarios solamente tendrán acceso a ver las fotos.
* Usuarios registrados: Este tipo de usuario puede ver las fotos, subir nuevas fotos, modificar las existentes al igual que eliminarlas.

## 2.3 Restricciones

Al tratarse de una aplicación web, se requiere un ordenador o un dispositivo móvil con un navegador convencional y una conexión a Internet básica.

## 2.4 Supuestos y dependencias

La aplicación desarrollada trabaja al margen de cualquier hardware o software ofreciendo así un soporte multiplataforma. La única dependencia importante que podemos encontrar está relacionada con el servidor web donde se encuentre alojado nuestro portal, el cual ha de ser capaz de soportar NodeJS y MongoDB.

## 2.5 Requisitos específicos

### 2.5.1 Requerimientos de interfaces externos

Al tratarse de una aplicación web, se podrá visualizar sobre cualquier sistema operativo.

La aplicación funcionará en cualquier máquina con un navegador web y conexión a Internet.

Las comunicaciones se efectuarán siguiendo el protocolo HTTP.

Debido a la metodología empleada en el diseño de la aplicación, el portal web tiene un rendimiento del 100% independientemente del hardware y software utilizado. Los únicos factores que pueden influir son las velocidades de conexión del cliente y del servidor. En cuanto al servidor, actualmente la mayoría cuentan con velocidades de subida más que suficientes para nuestras necesidades.

Respecto al cliente, las velocidades ofrecidas actualmente por los ISP también son más que suficientes para la correcta transmisión de datos entre el cliente y el servidor.

Otro aspecto que hemos tratado de cuidar especialmente es la resolución del monitor. La mejor resolución para visualizar nuestra aplicación es 1920x1080, pudiéndose visualizar, sin embargo, el contenido de igual modo con resoluciones inferiores debido a que la página web ha sido creada de manera responsive.

Se han intentado cumplir los estándares de cualquier web con acceso seguro, creando un sistema de autenticación para que nadie pueda acceder a una zona de la web a la que no tiene permiso de acceso. El idioma elegido para la presentación de las páginas ha sido el castellano.

Al tratarse de una aplicación web no se requiere un hardware específico. El servidor que albergará la base de datos del sistema deberá permanecer conectado a Internet las 24 horas, puesto que este host será quien atienda las peticiones de lectura y escritura de los usuarios que accedan a la intranet.

La aplicación ha sido diseñada con tecnología libre, luego podrá ser soportada por cualquier plataforma y sistema operativo. Por lo mismo, se podrá acceder a ella desde cualquier navegador o dispositivo movil.

# 3.Tecnologías utilizadas en el desarrollo del proyecto

## 3.1HTML y XHTML

HTML, siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>).

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin animo de lucro llamado World Wide Web Consortium, más conocido como W3C. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de forma muy similar en cualquier navegador de cualquier sistema operativo. El propio W3C define el lenguaje HTML como "un lenguaje reconocido universalmente y que permite publicar información de forma global".

Por convención, los archivos de formato HTML usan la extensión .htm o .html.

XHTML es el sucesor de HTML. XHTML, acrónimo inglés de eXtensible Hypertext Markup Language (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML.

XHTML tiene como objetivo avanzar en el proyecto del W3C de lograr una web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo y JavaScript su aspecto y diseño en distintos medios.

## 3.2 CSS

Las hojas de estilo en cascada (Cascading Style Sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C es el encargado de formular la especificación de las hojas de estilo que servirá de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. La información de estilo puede ser adjuntada tanto como un documento separado o en el mismo documento HTML. En este último podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo "style".

Las ventajas de utilizar CSS (u otro lenguaje de estilo) son:

* Control centralizado de la presentación de un sitio web completo, con lo que se agiliza de forma considerable la actualización del mismo.
* Los navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web remoto, con lo que aumenta considerablemente la accesibilidad.
* Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre.
* El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

## 3.3JavaScript

JavaScript es un lenguaje interpretado utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java. Sin embargo, al contrario que Java, JavaScript no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de herencia. Es más bien un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

Todos los navegadores interpretan el código JavaScript integrado dentro de las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM (Modelo de Objetos del Documento). JavaScript se ejecuta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

# 4.Documentación

La función save guarda datos en el modelo correspondiente de la base de datos.

save: (req, res) => {

// Recoger parametros por post

var params = req.body;

// Validar datos (validator)

try{

var validate\_nick = !validator.isEmpty(params.nick);

var validate\_email = !validator.isEmpty(params.email);

var validate\_pass = !validator.isEmpty(params.pass);

}catch(err){

return res.status(200).send({

status: 'error',

message: 'faltan datos'

});

}

if (validate\_nick && validate\_email && validate\_pass){

// Crear el objeto a guardar

var user = new User();

// Asignar valores

user.nick = params.nick;

user.email = params.email;

user.pass = params.pass;

user.idCou = params.idCou;

if(params.pfp){

user.pfp = params.pfp;

}else{

user.pfp = null;

}

// Guardar el objeto

user.save((err, userStored) => {

if (err || !userStored){

return res.status(404).send({

status: 'error',

message: 'El usuario no se ha guardado'

});

}

// Devolver una respuesta

return res.status(200).send({

status: 'success',

user: userStored

});

});

}else{

return res.status(200).send({

status: 'error',

message: 'Los datos no son válidos'

});

}

},

La función getXs devuelve todos los objetos del modelo correspondiente.

getUsers: (req, res) => {

var query = User.find({});

var last = req.params.last;

if(last || last != undefined){

query.limit(5);

}

// Find

query.sort('-\_id').exec((err, users) => {

if (err){

return res.status(500).send({

status: 'error',

message: 'Error al devolver los datos'

});

}

if(!users){

return res.status(404).send({

status: 'success',

message: 'No hay usuarios para mostrar'

});

}

return res.status(200).send({

status: 'success',

users

});

});

},

La función getXById devuelve el objeto del modelo correspondiente al Id que se manda.

getUserById: (req, res) => {

// Recoger el id de la url

var userId = req.params.id;

// Comprobar que existe

if(!userId || userId == null){

return res.status(404).send({

status: 'error',

message: 'No existe el usuario'

});

}

// Buscar el usuario

User.findById(userId, (err, user) => {

if(err || !user){

return res.status(404).send({

status: 'error',

message: 'No existe el usuario'

});

}

// Devolverlo en json

return res.status(200).send({

status: 'success',

user

});

})

},

La función getX devuelve el objeto del modelo correspondiente al parámetro que se manda. En este por ejemplo se devuelve el usuario por el Nick.

getUser: (req, res) => {

// Recoger el id de la url

var userNick = req.params.nick;

// Comprobar que existe

if(!userNick || userNick == null){

return res.status(404).send({

status: 'error',

message: 'No existe el usuario'

});

}

// Buscar el usuario

User.findOne({nick: userNick}, (err, user) => {

if(err || !user){

return res.status(404).send({

status: 'error',

message: 'No existe el usuario'

});

}

// Devolverlo en json

return res.status(200).send({

status: 'success',

user

});

})

},

La funcion de login autentica el usuario y lo deja loguearse a la página o no dependiendo de si se encuentran sus datos validados en la base de datos y se devuelve un jwt si el resultado es positivo.

login: (req, res, next) => {

User.find({ nick: req.body.nick })

.exec()

.then(user => {

if (user.length < 1) {

return res.status(401).json({

message: "Auth failed"

});

}

if (req.body.pass == user[0].pass){

const token = jwt.sign(

{

nick: user[0].nick,

userId: user[0].\_id

},

process.env.JWT\_KEY,

{

expiresIn: "1h"

}

);

return res.status(200).json({

message: "Auth successful",

token: token

});

} else {

return res.status(401).json({

message: "Auth failed"

});

}

});

},

La función update modifica el objeto del modelo correspondiente.

update: (req, res) => {

// Recoger el id del usuario por la url

var userId = req.params.id;

// Recoger los datos que llegan por put

var params = req.body;

// Validar datos

try{

var validate\_nick = !validator.isEmpty(params.nick);

}catch(err){

return res.status(200).send({

status: 'error',

message: 'Faltan datos por enviar'

});

}

if(validate\_nick){

// Find and update

User.findOneAndUpdate({\_id: userId}, params, {new:true}, (err, userUpdated) => {

if(err){

return res.status(500).send({

status: 'error',

message: 'Error al actualizar'

});

}

if(!userUpdated){

return res.status(404).send({

status: 'error',

message: 'No existe el usuario'

});

}

return res.status(200).send({

status: 'success',

user: userUpdated

});

});

}else{

// Devolver respuesta

return res.status(200).send({

status: 'error',

message: 'La validación no es correcta'

});

}

},

La función delete borra el objeto del modelo correspondiente

delete: (req, res) => {

// Recoger el id de la url

var userId = req.params.id;

// Find and delete

User.findOneAndDelete({\_id: userId}, (err, userRemoved) => {

if(err){

return res.status(500).send({

status: 'error',

message: 'Error al borrar'

});

}

if(!userRemoved){

return res.status(404).send({

status: 'error',

message: 'No se ha borrado el usuario, posiblemente no exista'

});

}

return res.status(200).send({

status: 'success',

user: userRemoved

});

});

},

La función upload sube una imagen a traves del middleware connect-multiparty

upload: (req, res) => {

// Recoger el fichero de la petición

var file\_name = 'Imagen no subida';

if(!req.files){

return res.status(404).send({

status: 'error',

message: file\_name

});

}

// Conseguir nombre y la extensión del archivo

var file\_path = req.files.file0.path;

var file\_split = file\_path.split('\\');

// Nombre del archivo

var file\_name = file\_split[2];

// Extensión del fichero

var extension\_split = file\_name.split('\.');

var file\_ext = extension\_split[1];

// Comprobar la extension, solo imagenes, si es valida borrar el fichero

if(file\_ext != 'png' && file\_ext != 'jpg' && file\_ext != 'jpeg' && file\_ext != 'gif'){

// borrar el archivo subido

fs.unlink(file\_path, (err) => {

return res.status(200).send({

status: 'error',

message: 'La extensión de la imagen no es válida'

});

});

}else{

// Si todo es valido, sacando id de la url

var userId = req.params.id;

if(userId){

// Buscar el usuario, asignarle el nombre de la imagen y actualizarlo

User.findOneAndUpdate({\_id: userId}, {pfp: file\_name}, {new:true}, (err, userUpdated) => {

if(err || !userUpdated){

return res.status(200).send({

status: 'error',

message: 'Error al guardar la imagen'

});

}

return res.status(200).send({

status: 'success',

user: userUpdated

});

});

}else{

return res.status(200).send({

status: 'success',

pfp: file\_name

});

}

}

}, // end upload file

La función get image devuelve la Imagen que se pasa por parametro.

getImage: (req, res) => {

var file = req.params.getImage;

var path\_file = './upload/images/'+file;

fs.exists(path\_file, (exists) => {

if(exists){

return res.sendFile(path.resolve(path\_file));

}else{

console.log(path\_file);

return res.status(404).send({

status: 'error',

message: 'La imagen no existe'

});

}

});

},

La función search te devuelve el objeto que corresponda con el string que se le pasa por parametro.

search: (req, res) => {

// Sacar el string a buscar

var searchString = req.params.search;

// Find or

User.find({ "$or": [

{ "nick": { "$regex": searchString, "$options": "i"}}

]})

.exec((err, users) => {

if(err){

return res.status(500).send({

status: 'error',

message: 'Error en la petición'

});

}

if(!users || users.length <= 0){

return res.status(404).send({

status: 'error',

message: 'No hay usuarios que coincidan con tu busqueda'

});

}

return res.status(200).send({

status: 'success',

users

});

});

}

Estas funciones usan el middleware de mongoose para conectarse a la base de datos de MongoDB

mongoose.set('useFindAndModify', false);

mongoose.Promise = global.Promise;

mongoose.connect(MONGODB\_URI, { useUnifiedTopology : true, useNewUrlParser : true, useCreateIndex: true })

.then(() => {

console.log('Conexión a la base de datos correcta');

// Crear servidor y escuchar peticiones HTTP

app.listen(port, () => {

console.log('Servidor corriendo en http://localhost:'+ port);

});

});

// Cargar modulos de node para crear el servidor

var express = require('express');

var bodyParser = require('body-parser');

const { restart } = require('nodemon');

// Ejecutar express (http)

var app = express();

// Cargar ficheros rutas

var routes = require('./routes/routes');

// Middlewares

app.use(bodyParser.urlencoded({extended:false}));

app.use(bodyParser.json());

// CORS

app.use((req, res, next) => {

res.header('Access-Control-Allow-Origin', '\*');

res.header('Access-Control-Allow-Headers', 'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type, Accept, Access-Control-Allow-Request-Method');

res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, DELETE');

res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');

next();

});

// Añadir prefijos a rutas / Cargar rutas

app.use('/api', routes);