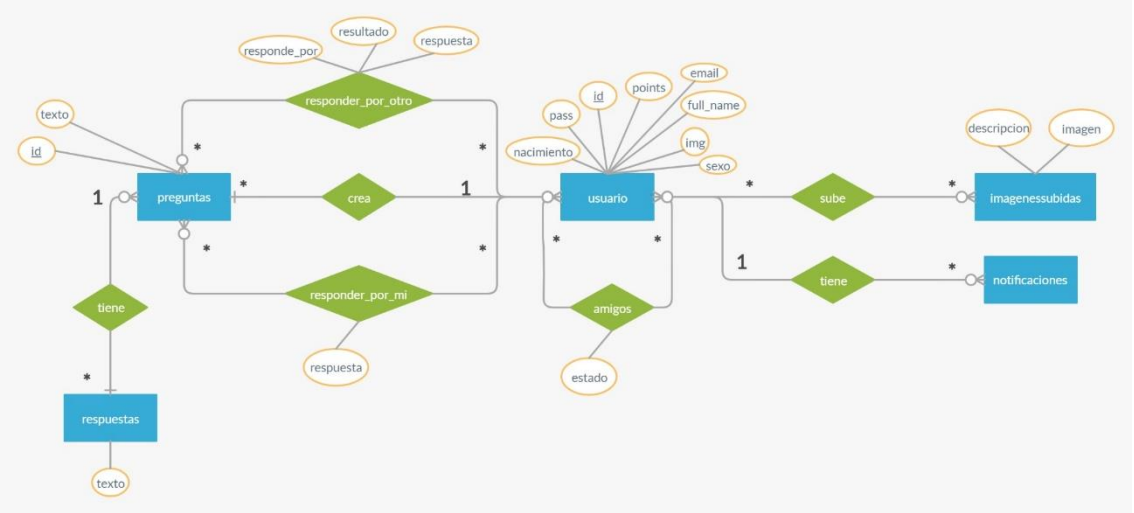
Presentación Facebluff

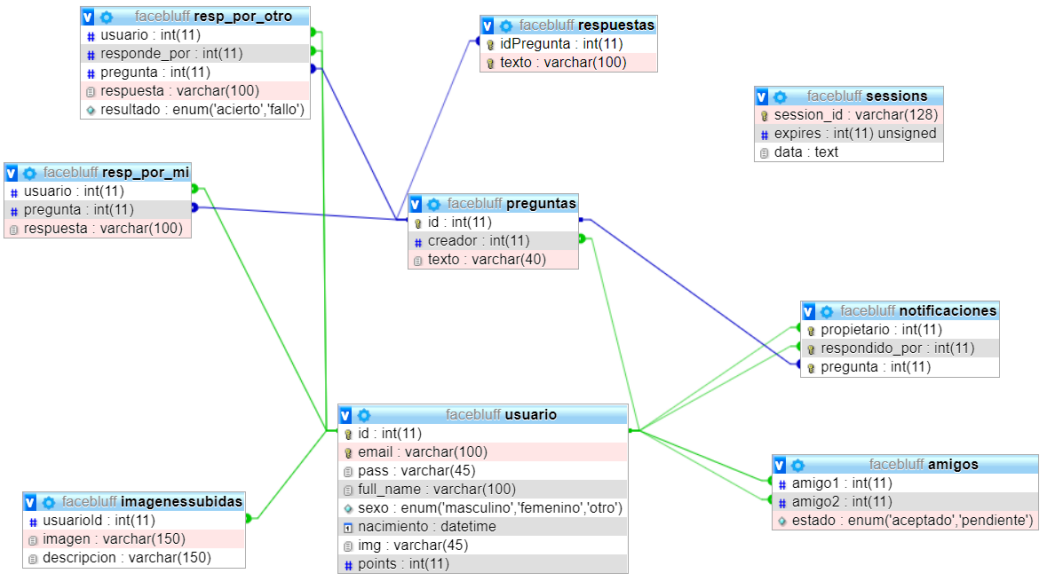
Índice

Diseño de la base de datos:

- Modelo entidad-relación:



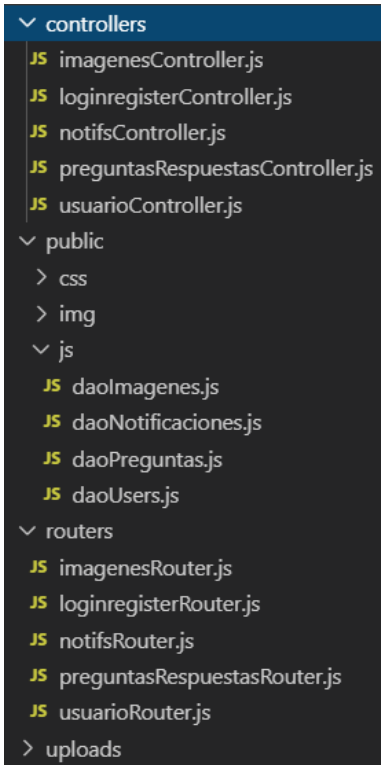
- Modelo relacional:

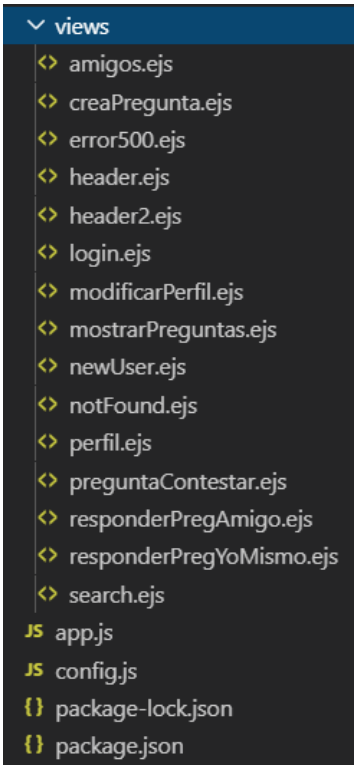


Estructura de la aplicación:

En el directorio raíz encontraremos la carpeta de controllers, la carpeta de routers, la carpeta de vistas, la carpeta de descargas donde se encuentran las imágenes, el archivo de configuración de la base de datos con el puerto que escucha el servidor, el fichero app.js que es el ejecutor de la aplicación web y la carpeta public, en la que se encuentra la carpeta css con los css´s, otra carpeta de imágenes de los css y la carpeta js con los daos.

- Árbol de directorios





Listado de rutas gestionadas por el servidor:

- notifsRouter.js

  - Todas las rutas llaman a las funciones de notifsController.js

Nombre ruta	Tipo	Descripción
/eliminaNotif	POST	Llama a la función eliminaNotif que se encarga de eliminar todas las notificaciones de un usuario dado.

- imagenesRouter.js

    - Todas las rutas llaman a las funciones de imagenesController.js

Nombre ruta	Tipo	Descripción
/subirFoto	POST	Llama a la función subirImagenPost que se encarga de subir una imagen dado un usuario, una imagen elegida por el usuario y una descripción de la imagen.

- loginregisterRouter.js

    - Todas las rutas llaman a las funciones de loginregisterController.js

Nombre ruta	Tipo	Descripción
/login	GET	Llama a la función loginGET que se encarga de hacer un render de la pantalla de login con el mensaje en nulo.
/login	POST	Llama a la función loginPOST que comprueba si el usuario está logueado o ha introducido los datos correctos para redireccionarle a la pantalla de perfil, en caso contrario volverá al login con un error.
/registrar	GET	Llama a la función registrarGET que hace el render de la pantalla newUser con el error en nulo.
/registrar	POST	Llama a la función registrarPOST que comprueba que todos los datos del formulario introducidos son correctos para así poder crear el usuario. Una vez creado redireccionará la página al perfil, y en caso de haber algún error volverá al formulario anterior con el error correspondiente.

- preguntasRespuestasRouter.js

    - Todas las rutas llaman a las funciones de preguntasRespuestasController.js

Nombre ruta	Tipo	Descripción
/preguntas	GET	Llama a la función muestraPreguntasGET que hace el render de la pantalla con las preguntas aleatorias existentes en la base de datos.
/crearPregunta	GET	Llama a la función creaPreguntaGET que se encarga de hacer el render del formulario para crear una pregunta.

/crearPregunta	POST	Llama a la función creaPreguntaPOST que crea la pregunta introducida con sus respuestas en la base de datos.
/pregunta/:id	GET	Llama a la función preguntaSeleccionadaGET devolviendo con un render la pantalla preguntaContestar dando la oportunidad al usuario de contestar las preguntas de sus amigos en el caso de que hayan contestado dicha pregunta.
/pregunta/contestaryo/:id	GET	Llama a la función preguntaContestrarYoMismoGET la cual, dado el id de la pregunta, devuelve la vista responderPregYoMismo para que puedas responder la pregunta.
/pregunta/contestaryo/:id	POST	Llama a la función responderPOST que se encarga de subir la respuesta elegida por el usuario a la base de datos para que sus amigos puedan adivinar así cual fue su respuesta seleccionada.
/pregunta/contestarporotro	POST	Llama a la función responderPorOtro que se encarga de hacer el render de responderPregAmigo, mostrando así en la vista las posibles respuestas que tu amigo pudo elegir.
/pregunta/contestadaporotro	POST	Llama a la función respondidaPorOtro que comprueba si la respuesta elegida por el usuario corresponde con la respuesta elegida por el amigo y así poder ganar puntos.

- usuarioRouter.js

    - Todas las rutas llaman a las funciones de usuarioController.js

Nombre ruta	Tipo	Descripción
/perfil	GET	Llama a la función perfil que se encarga de mostrar el perfil completo, años, foto de perfil, nombre completo, sexo, puntos y fotos subidas por el usuario junto con su descripción.
/perfil/:id	GET	Llama a la función perfil mostrando el perfil completo del id del usuario pasado.
/imagen/:id	GET	Llama a la función getImagen que devuelve la imagen del usuario.
/logout	GET	Llama a la función logOut que desconecta al usuario, borrando la sesión de este y redireccionando la página al login inicial.
/amigos	GET	Llama a la función amigos que se encarga de mostrar la vista de amigos, tanto de amigos como de solicitudes de amistad pendientes.
/aceptar/:id	GET	Llama a la función aceptarAmigo que añade a la base de datos el amigo aceptado.
/rechazar/:id	GET	Llama a la función rechazarAmigo borrando así el amigo de la base de datos del usuario.

/modificarPerfil	GET	Llama a la función modificarPerfilGET que devuelve la vista modificarPerfil con los datos actuales del usuario para que este pueda modificarlos a su antojo.
/modificarPerfil	POST	Llama a la función modificarPerfilPOST que comprueba que los datos modificados son correctos antes de subirlos a la base de datos, si son correctos los modificará y redireccionará al perfil del usuario.
/search	POST	Llama a la función search que busca a un usuario por su nombre. Una vez encontrado lo devolverá en la vista search.
/solicitar/:id	GET	Llama a la función solicitar que se encarga de crear una petición de amistad entre el usuario y otro usuario, una vez creada la petición se redireccionará a la vista de amigos.

Implementación de la sesión para un usuario logueado:

Al loguearse el usuario, se comprueba si es correcto el usuario y contraseña introducidos, si es así, se inicializan los datos de sesión, el id del usuario, su imagen y los puntos, que serán utilizados en todo momento mientras el usuario está logueado.

```
function loginPOST(request, response) {
    daoUser.isUserCorrect(request.body.name, request.body.pass, function(error, usuario) {
        if (error) {
            response.status(200);
            response.render("login", { errorMsg: "Error" });
        } else {
            if (usuario) {
                request.session.userId = usuario.id;
                request.session.img = usuario.img;
                request.session.puntos = usuario.puntos;
                response.redirect("perfil")
            } else {
                response.status(200);
                response.render("login", { errorMsg: "Usuario o contraseña incorrectos" });
            }
        }
    });
}
```

Al registrarse un usuario pasa igual que en el caso anterior, si los datos son introducidos correctamente en la base de datos, los datos usados para añadir el usuario en la base de datos se usan para inicializar los datos de sesión.

```
daoUser.addUser(request.body.name, request.body.pass,
    request.body.full_name, request.body.sexo, nacimiento, img,
    function(error, data) {
        if (error) {
            response.status(200);
            response.render("newUser", { errorMsg: error.message })
        } else {
            //response.cookie("userId", id)
            request.session.userId = data.id;
            request.session.img = data.img;
            request.session.puntos = data.puntos;
            response.redirect("/perfil");
        }
    });
```

Restricción de acceso a las rutas para los usuarios no logueados:

- MiddlewareControlAcceso

    - Usamos un middleware de control de acceso para impedir el acceso de usuarios sin haberse logueado o sin ni si quiera haberse registrado. Este se encuentra en el router loginregisterRouter.js y se ejecutará siempre lo primero, comprobando si el url que recibe es el login o el registrar y si el usuario estaba o no estaba ya logueado.

```
loginRegisterRouter.use(loginRegisterController.middlewareControlAcceso);
```

```
function middlewareControlAcceso(request, response, next) {
    if (request.url == "/login" || request.url == "/registrar") {
        if (request.session.userId !== undefined) {
            response.redirect("/perfil");
        } else {
            next();
        }
    } else {
        if (request.session.userId !== undefined) {
            next();
        } else {
            response.redirect("/login");
        }
    }
}
```

Gestión de los errores 404 y 500:

Ambos se encuentran en el app.js

```
// Cadena de middlewares
app.use(middlewareNotFoundError);
app.use(middlewareServerError);
```

- Error 404

```
function middlewareNotFoundError(request, response) {
    response.status(404);
    // envío de página 404
    response.render("notFound.ejs", {
        url: request.url
    });
}
```

- Error 500

```
function middlewareServerError(error, request, response, next) {
    response.status(500);
    // envío de página 500
    response.render("error500.ejs", {
        mensaje: error.message,
        pila: error.stack
    });
    response.end()
}
```