

2019

FaceToFace

*Autores: Ana Laura Corral Descargue y Guillermo Delgado Yepes*

Practica de aplicaciones web

FaceToFace

Contenido

[1. Diseño de la base de datos 3](#_Toc26916224)

[2. Estructura de la aplicación 4](#_Toc26916225)

[3. Listado de las rutas gestionadas por el servidor 7](#_Toc26916226)

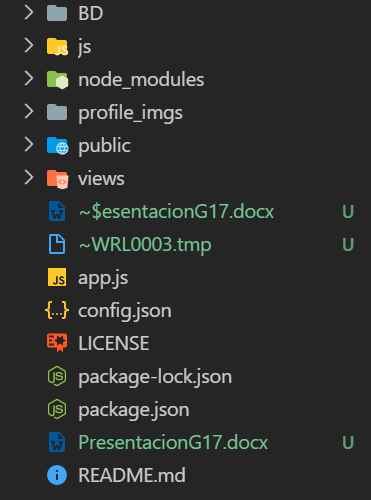
[4. Implementación de la sesión para un usuario logueado 9](#_Toc26916227)

[5. Restricción de acceso a las rutas para el usuario no logueado 9](#_Toc26916228)

[6. Gestión de los errores 404 y 500 10](#_Toc26916229)

# Diseño de la base de datos

# Estructura de la aplicación



En la raíz del proyecto tendremos:

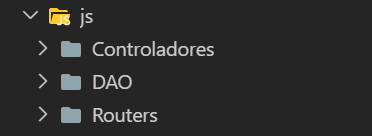
* App.js: quien inicia el servidor, contiene los módulos necesarios para la página web, las rutas estáticas que usaremos y el manejador de los errores 404 y 500
* Config.json: configuración de la base de datos y puerto
* Package.json archivo de configuración de node.js

Y en las diferentes carpetas tendremos:

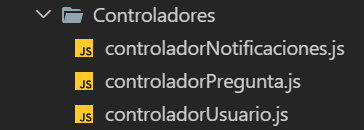
* Base de dato: script para importar la base de datos



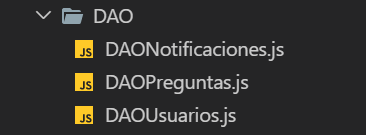
* Js: tendremos los archivos .js



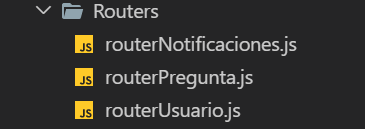
* Controlador: realiza la parte lógica de la página web, donde están todas funciones



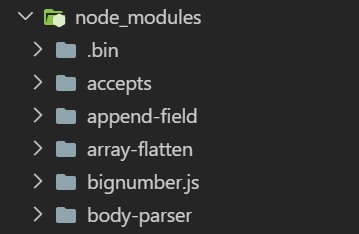
* DAO: están las querys con acceso a la base de datos



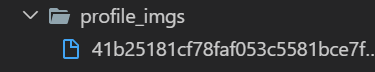
* Routers: tendremos las rutas para los diferentes middlewares donde llamaremos a las diferentes funciones del controlador asociadas a las rutas



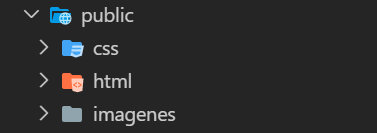
* Node\_modules: se almacenará los módulos necesarios para el correcto funcionamiento de la aplicación



* Profile\_imgs: se guardarán las imágenes subidas por los diferentes usuarios



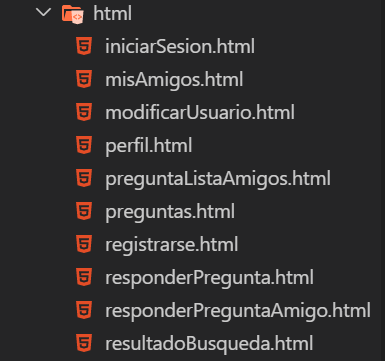
* Public: tendremos los html, css e imágenes



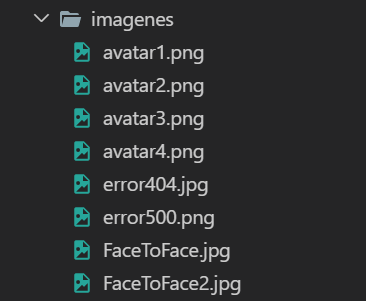
* CSS: le dará el estilo a nuestra página web



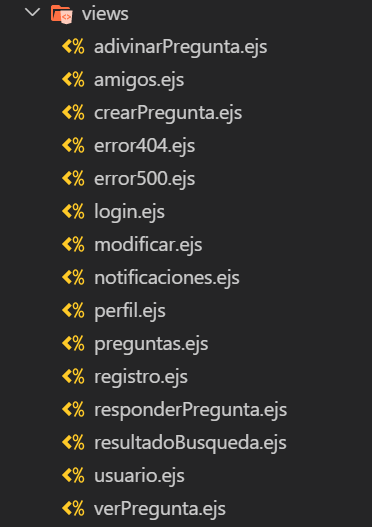
* HTML: formato estático y provisional de como iban a ser las diferentes ventanas de la página web



* Imágenes: contiene imágenes de prueba y imágenes estáticas como el logo que nunca cambian



* Views: contiene todos los ejs de las diferentes ventanas dinámicas de la página web



# Listado de las rutas gestionadas por el servidor

* routerNotificaciones.js:
  + router.get("/leido/:id", controladorNotificaciones.getMarcarLeido);

Manda el id de la pregunta para marcarlo como leido

* + router.get("/leido/", controladorNotificaciones.getMarcarTodas);

Marca todas las notificaciones como leídas

* routerPregunta.js
  + router.get("/preguntas", controladorPregunta.getPreguntas);

Genera la vista donde aparecen 5 preguntas aleatorias

* + router.get("/crearPregunta", controladorPregunta.getCrearPregunta);

Genera la vista para rellenar el formulario de crear una pregunta

* + router.post("/insertarRespuestas", controladorPregunta.postInsertarRespuestas);

Inserta en la base de datos las respuestas de una pregunta

* + router.post("/crearPregunta", controladorPregunta.postCrearPregunta);

Crea una pregunta

* + router.get("/verPregunta/:id/:pregunta", controladorPregunta.getVerPregunta);

Genera la vista de la pregunta donde necesitamos mandar el id de la pregunta para saber cual tenemos que mostrar y el titulo de la pregunta para mostrarla por pantalla

* + router.get("/responder/:id", controladorPregunta.getResponderPregunta

Genera la vista para responder la pregunta del id correspondiente

* + router.post("/responder", controladorPregunta.postResponderPregunta);

Se responde a la pregunta

* + router.post("/addRespuesta", controladorPregunta.postAddRespuesta);

Añade una nueva respuesta a una pregunta

* + router.get("/adivinar/:pregunta/:amigo/:nombre", controladorPregunta.getAdivinarPregunta);

Genera la vista para adivinar la respuesta de un amigo, para eso necesitamos mandar el titulo de la pregunta, el id del aimigo que voy adivinar la pregunta y el nombre del amigo

* + router.post("/adivinar", controladorPregunta.postAdivinarPregunta);

Se marca la pregunta como adivinada o no

* routerUsuario.js:
  + router.get("/registro", controladorUsuario.getRegistro);

Genera la vista para registrarse

* + router.post("/registro", multerFactory.single("foto"), controladorUsuario.postRegistro);

Crea un nuevo usuario subiendo una foto de perfil

* + router.get("/login", controladorUsuario.getLogin);

Genera la vista de login

* + router.post("/login", controladorUsuario.postLogin);

Comprueba que el usuario y contraseña si son correctos y lo manda a la vista de perfil

* + router.get("/logout", controladorUsuario.getLogout);

Nos desconectamos de nuestro usuario

* + router.get("/perfil", controladorUsuario.getPerfil);

Nos genera la vista de perfil con la información del usuario

* + router.get("/imagenUsuario/:imagen", controladorUsuario.getImagenUsuario);

Se observa la imagen del usuario elegida

* + router.get("/imagenUsuario/", controladorUsuario.getImagenPorDefecto);

Se observa una foto de perfil por defecto

* + router.get("/modificar", controladorUsuario.getModificar);

Genera la vista para modificar los datos del usuario

* + router.post("/modificar", multerFactory.single("foto"), controladorUsuario.postModificar);

Modifica los datos del usuario

* + router.get("/amigos", controladorUsuario.getAmigos);

Genera la vista con los amigos aceptados

* + router.post("/buscar", controladorUsuario.postBuscar);

Realiza la búsqueda de los usuarios que no tenemos agregados como amigos

* + router.get("/verUsuario/:correo/:amigo", controladorUsuario.getPerfilUsuario);

Genera la vista de la información del amigo para eso necesitamos saber quienes tenemos agregado como amigo

* + router.get("/solicitar/:correo", controladorUsuario.getPeticion);

Genera la vista para enviar una solicitud de amistad para eso necesitamos nuestro correo

* + router.get("/aceptar/:correo", controladorUsuario.getAceptar);

Acepta un usuario

* + router.get("/rechazar/:correo", controladorUsuario.getRechazar);

Rechaza un usuario

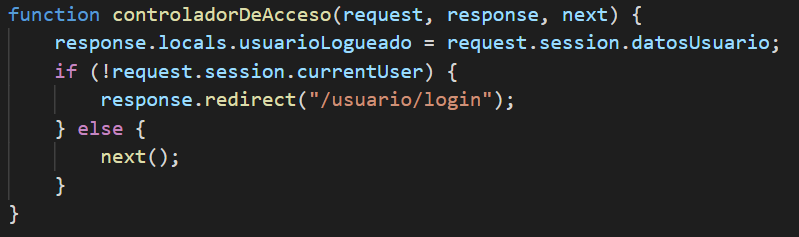
* + router.get("/notificaciones", controladorUsuario.getNotificaciones);

Genera las notificaciones

* + router.post("/foto", multerFactory.single("foto"), controladorUsuario.postSubirFoto);

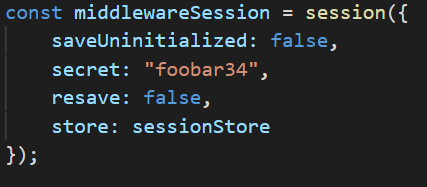
Sube la foto

# Implementación de la sesión para un usuario logueado



Una vez que esta logueado comprobamos que el usuario que se registro es igual a lo que tenemos almacenado en la base de datos de session

# Restricción de acceso a las rutas para el usuario no logueado



Para evitar que acceda a rutas a las cuales necesita estar logueado para acceder a ellas y evitar este tipo de problema, se utiliza **request.session.currentUser** para saber si esta logueado o no

# Gestión de los errores 404 y 500