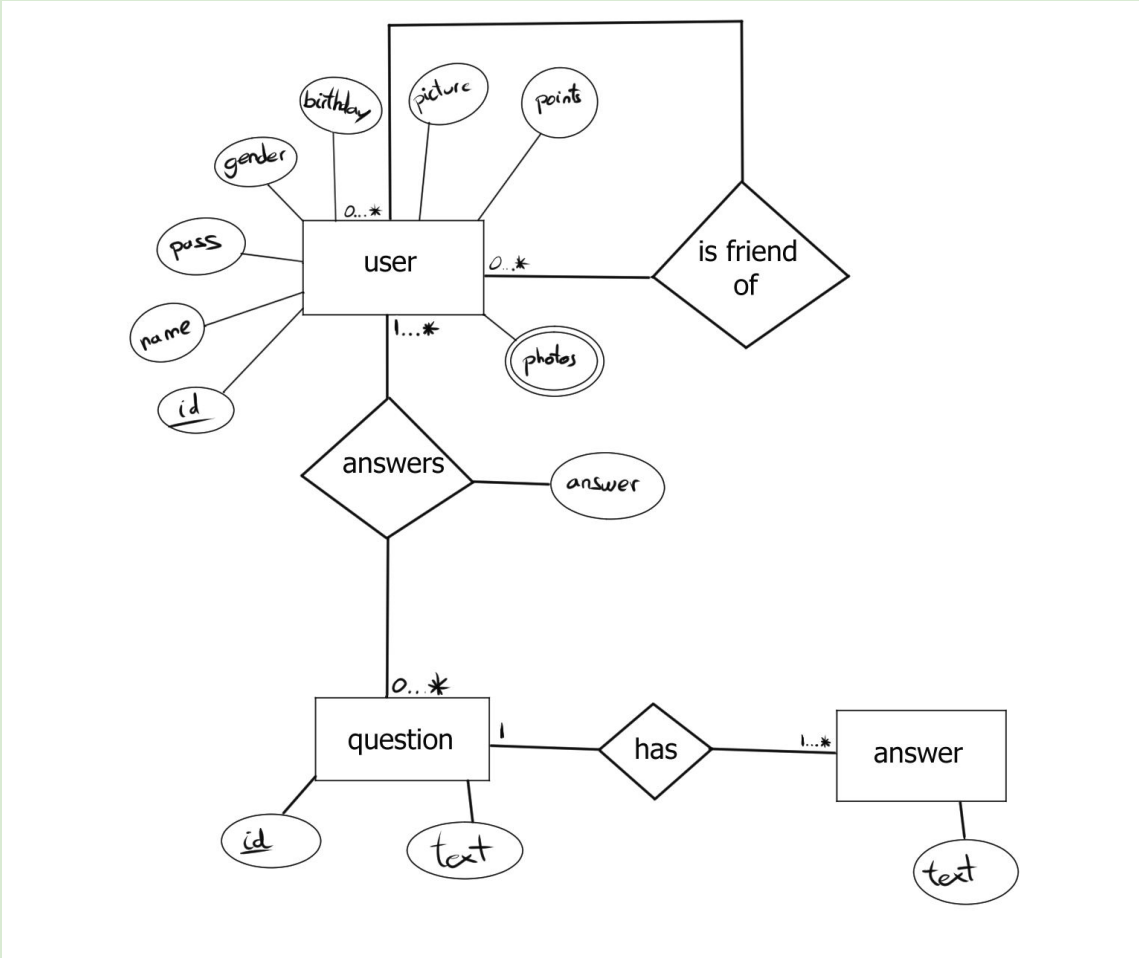
# Práctica 1: Facebluff

Adrián Ogáyar Sánchez
Maikel Jesús Spranger Hierro

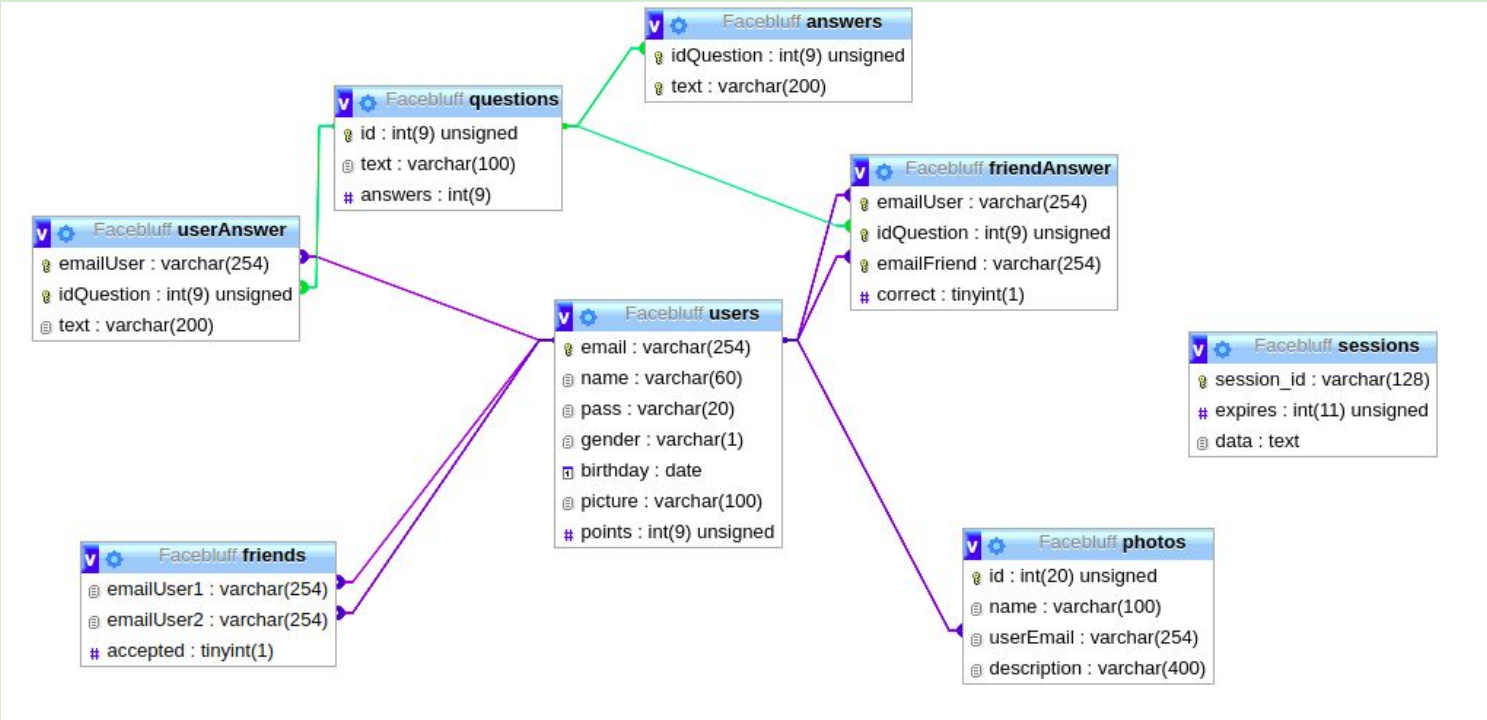
# Diseño de la base de datos

## Modelo entidad-relación



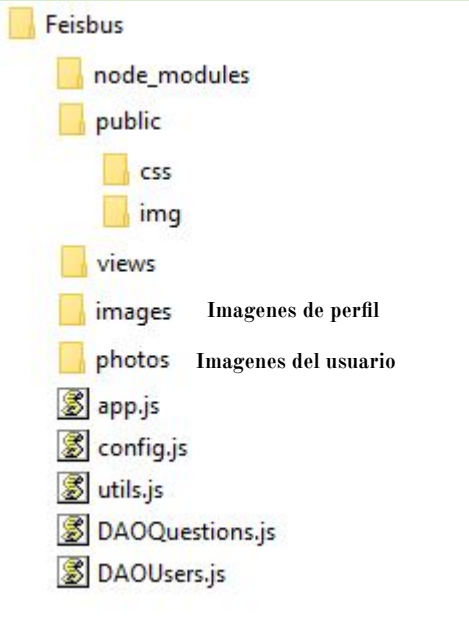
# Diseño de la base de datos

## Modelo relacional

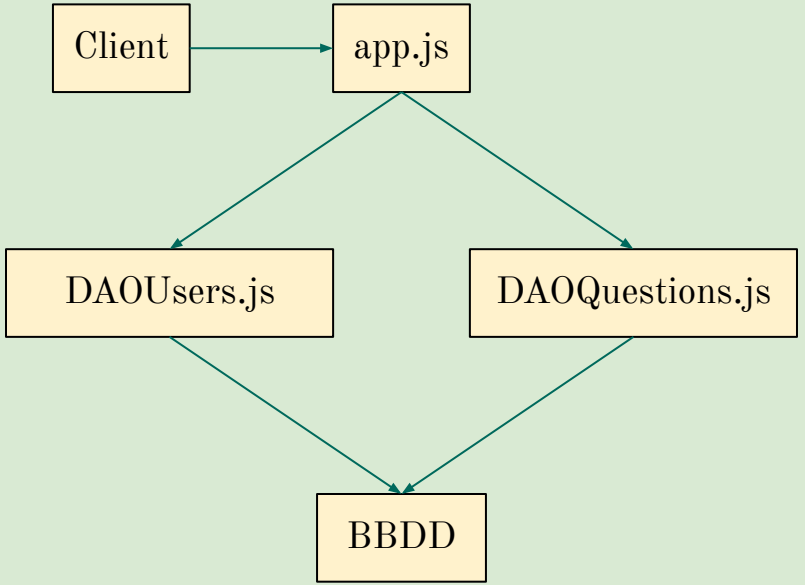


# Estructura de la aplicación

Estructura de ficheros




Estructura de la aplicación



## Listado de las rutas gestionadas por el servidor

**Nota**: Todas las operaciones de los DAO envían un error. Este error se muestra mediante un *console.log*, esta ejecución se omite en la explicación de las rutas por simplicidad. También se omite la renderización de las plantillas, que siempre reciben los datos mencionados en la ruta y, por omisión, los manejadores son get().

"/": Ruta principal de la página, redirige al login.

"/login": Login de la página. Muestra un formulario para introducir email y contraseña.

post("/login"): Recoge el email y la contraseña enviados desde el login y comprueba si son correctos. En caso afirmativo redirige al perfil, sino muestra un mensaje de error.

"/logout": Elimina la sesión, devolviendo al login.

"/signin": Muestra un formulario para la creación de un usuario.

## Listado de las rutas gestionadas por el servidor

“/userImage”: Busca la imagen del usuario logueado y si la encuentra la envía al cliente, en caso contrario devuelve la imagen por defecto (smile.jpg).
Existe otra variante de la ruta con parámetro (userImage/:email), en ese caso busca el la imagen del usuario con ese email y realiza los pasos anteriores.

“/friends”: Obtiene los amigos del usuario logueado y la lista de peticiones de amistad para mostrarlos al usuario.

post(“/createUser”): Recoge los datos enviados desde el signin, si el email ya existe muestra un error en la página de signin, en caso contrario lo crea, guarda el email en la sesión y redirige al perfil del usuario.

“/profile”: Obtiene los datos del usuario logueado y sus fotos para mostrarlos en la página. Se encarga de convertir los caracteres M, F y O en “Male”, “Female” y “Other”, además de calcular la edad a partir de la fecha de nacimiento.
Existe otra variante con parámetro (profile/:email) que obtiene los datos del usuario con ese email.

## Listado de las rutas gestionadas por el servidor

post(“/search”): Obtiene una lista con los usuarios que contienen la cadena introducida (recogida por el post). El DAO devuelve los usuarios y una variable que indica si ese usuario es amigo del usuario logueado.

“/questions”: Obtiene 5 preguntas aleatorias de la base de datos para mostrarlas al usuario.

“/newQuestion”: Renderiza la plantilla de crear pregunta.

post(“/createQuestion”): Crea la pregunta y, utilizando el id de la pregunta insertada, crea las respuestas introducidas por el usuario.

“/question/:id”: Obtiene la pregunta con el id correspondiente, la respuesta del usuario logueado (en caso de que exista) y las “suposiciones” del usuario de las respuestas de sus amigos.

“/answerQuestion/:id": Obtiene la pregunta con el id correspondiente y las respuestas asociadas a esa pregunta.

## Listado de las rutas gestionadas por el servidor

“/userAnswer/:id”: Comprueba si la respuesta es nueva para esa pregunta, si lo es, la asocia a la pregunta, lo sea o no también asocia la respuesta al usuario logueado y la pregunta con el parámetro id.

post(“/guess”): Obtiene la pregunta, los datos del amigo pasado por post, la respuesta de ese amigo a esa pregunta y una lista de respuestas elegidas aleatoriamente (con un número igual al de las respuestas iniciales a esa pregunta).

post(“/guessAnswer/:id”): Obtiene la respuesta  a la pregunta con el id pasado por parámetro del amigo pasado en el post, comprueba si esa respuesta coincide con la suposición del usuario y guarda en la base de datos el resultado. En caso de acierto aumenta en 50 los puntos del usuario.

“/updateProfile”: Obtiene los datos del usuario, cambiando el formato de la fecha de nacimiento a YYYY-MM-DD (al sacarlo de la base de datos tiene formato Date) para mostrarlo en el cliente.

## Listado de las rutas gestionadas por el servidor

post(“updateUser”): Hace un “login” para comprobar que la contraseña introducida es correcta, en caso positivo mete los datos en un objeto usuario, comprobando que se ha introducido nueva contraseña y/o imagen de perfil para saber si se deben cambiar, y actualiza los datos en la base de datos, en caso negativo muestra un mensaje de error en la plantilla de update.

post(“requestFriend”): Comprueba si el usuario logueado y el usuario enviado por post tienen una relación en la tabla friends, si no existe envia la petición al usuario del post, en caso afirmativo comprueba que el usuario que inició esa relación no es el usuario logueado, y si no lo es comprueba si el usuario del post ha rechazado la petición de amistad. Si la ha rechazado borra la petición de la tabla y crea una nueva petición de amistad.

post(“/acceptFriend”): Acepta la petición de amistad recibida por el usuario logueado y enviada por el usuario recogido por post.

post(“/ignoreFriend”): Ignora la petición de amistad, marcándolo en la base de datos (para que no se muestren más peticiones de ese usuario).

## Listado de las rutas gestionadas por el servidor

post(“/addPhoto”): Comprueba si el usuario logueado tiene al menos 100 puntos, si no es así, muestra un mensaje de aviso, en caso positivo mete los datos recibidos por post en un objeto y lo inserta en la base de datos, después reduce en 100 la puntuación del usuario logueado y le redirige al perfil.

(“/getPhoto/:name”): Comprueba si el parámetro name no es nulo, si lo es envía la imagen por defecto (smile.jpg), si no lo es busca el archivo en la carpeta “photos” y lo envía.

## Sesión para un usuario logueado

El servidor se encarga de mantener los datos de los usuarios que se loguean en la página para que este pueda acceder a las funciones de la misma.

Para ello cuando el usuario hace login, el servidor recoge el usuario y la contraseña, enviado mediante un post, comprueba si son correctos, y en caso positivo obtiene los puntos del usuario y los guarda junto al email en un *session*.

Además, el middleware *"middlewareCheckUser"* se ocupa de meter los datos en *respone.locals* par que así estén disponibles en las plantillas (principalmente *header.js*).

Cuando el usuario hace logout el servidor destruye la *session*.

# Restricción de acceso para usuarios no logueados

El mismo *middlewareCheckUser* que se ocupa de meter los datos en *response.locals*, también se encarga de comprobar si existe un email en la *session*, en caso afirmativo mete los datos en el response, pero en caso negativo redirige al login, impidiendo que cualquier usuario no registrado pueda acceder al resto de la aplicación.

Esto se hace colocando el *middleware* en todas las rutas donde no se quiere que el acceso sin autorización (todas excepto login y signIn).

# Gestión de errores 404 y 500

Para gestionar los errores 404 y 500 hemos creado dos funciones middleware:

middlewareNotFoundError.

middlewareServerError.

Estos middlewares gestionan el error 404 y 500 respectivamente renderizando una página que da un mensaje de error específico y que permite volver a la página principal (/friends).

Para la página 500 mostramos un mensaje de que ha habido un error y no los datos de la causa del error ya que desde el punto de vista del usuario esto no se podrá solucionar.