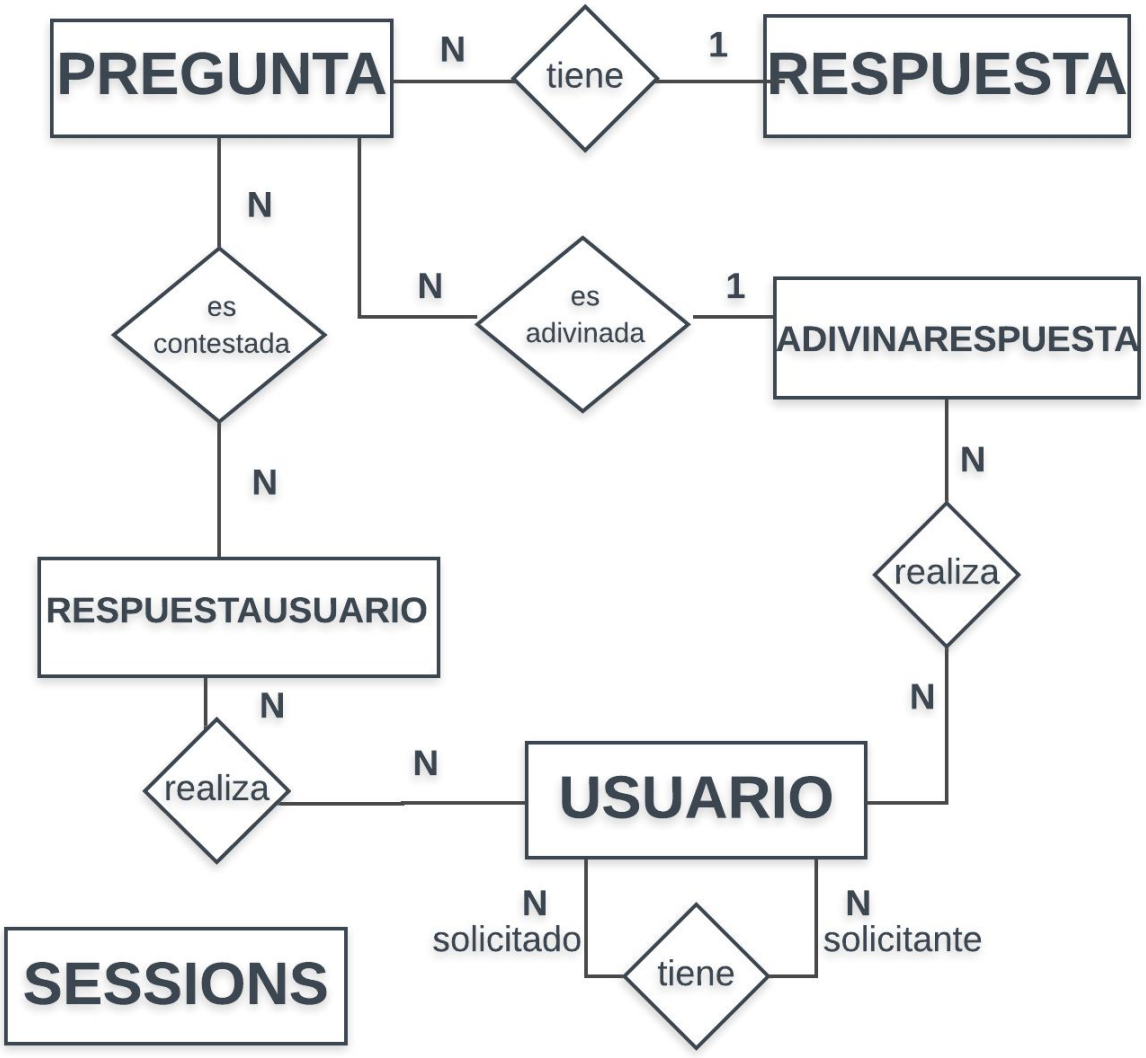
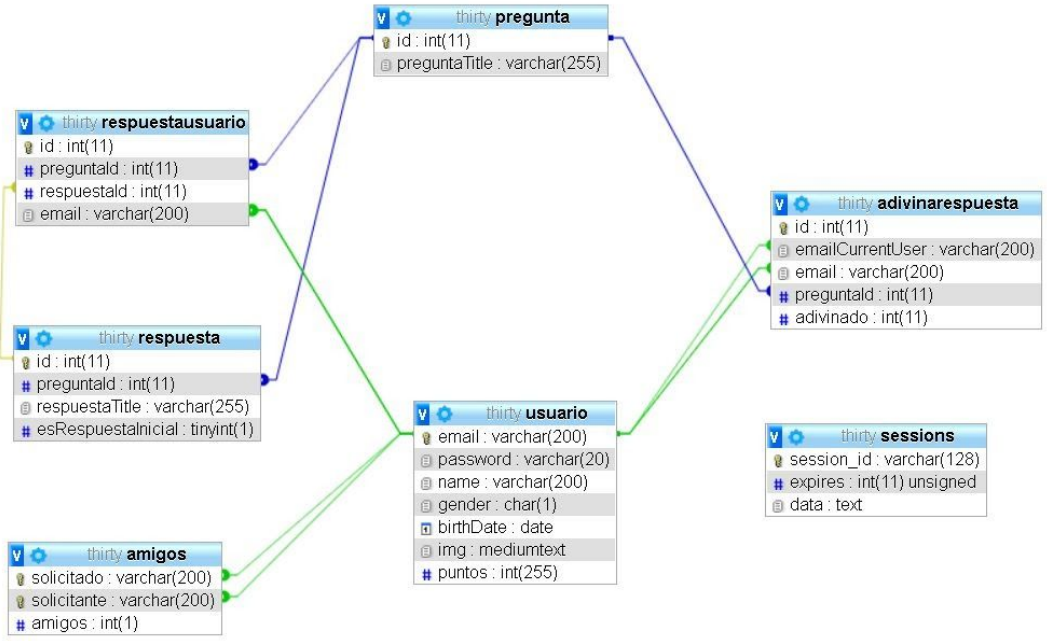
Práctica Facebluff

1. Diseño base de datos

a. Diseño entidad-relación



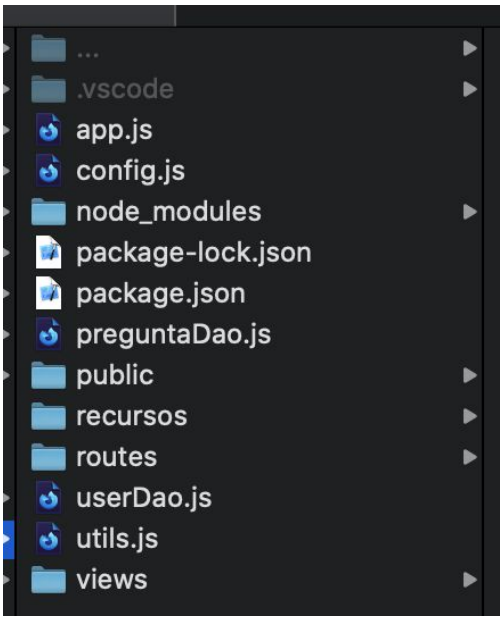
b. Diseño relacional

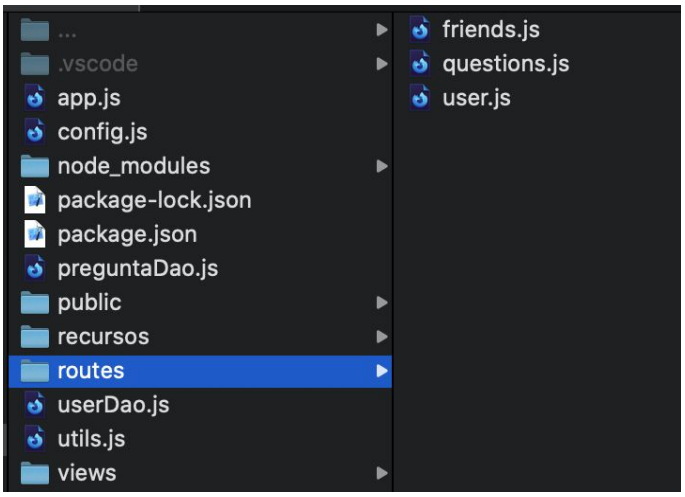


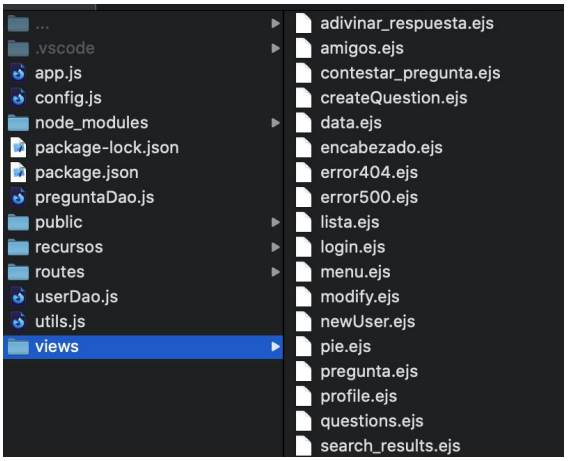
2. Estructura de la aplicación

La arquitectura de la aplicación se divide en:

Controlador (app.js): Este fichero se encarga de enlazar los routers.
Routers: Los podemos encontrar en la ruta /routes y son los ficheros, user.js, questions.js y friends.js. Estos ficheros contienen las rutas de las diferentes vistas que componen cada uno de los módulos de la aplicación.
Modelos: userDao.js y preguntaDao.js, estos módulos se encargan de gestionar el acceso a la base de datos. En el fichero userDao.js encontramos todas las peticiones relacionadas con los perfiles de los usuarios, mientras que en preguntaDao, encontramos todas las peticiones relacionadas con las preguntas.
/Views: En este módulo encontraremos todas las vistas de la aplicación.
/Public: En este módulo tenemos los ficheros relacionados con el diseño de la aplicación, así como las imágenes.
A continuación adjuntamos algunos capturas de la arquitectura de la aplicación.







3. Listado de rutas

- **“/”:** Lleva al usuario a la página principal, si el usuario está logueado se le redirigirá a su perfil, en caso contrario a la vista de login.
- **“/login”:** Muestra la vista de login.
- **“/register”:** Muestra la vista de registro.
- **“/profile”:** Muestra el perfil del usuario logueado.
- **“/modify”:** Muestra la vista de modificación de perfil del usuario.
- **“/logout”:** Cierra la sesión del usuario actual y muestra la página principal.
- **“/preguntas”:** Muestra un listado de preguntas.
- **“/crearPregunta”:** Muestra una vista donde un usuario puede crear una nueva pregunta con sus respuestas.
- **“/pregunta/:id”:** Muestra el detalle de una pregunta. Si el usuario ya ha respondido a esta pregunta se le notificará, en caso contrario se le dará la opción de contestar.
- **“/contestar_pregunta/:id”:** Muestra una pregunta y sus respuestas. Aquí el usuario podrá elegir la respuesta que le guste o en caso de no gustarle ninguna, añadir una nueva.
- **“/adivinar_respuesta/:id/:email”:** Muestra una pregunta con sus respuestas. Si el usuario acierta la respuesta de su amigo, ganará 50 puntos.
- **“/procesar_crear_pregunta”:** Procesa la creación de una pregunta, en caso de obtener todos los datos correctamente se delegan en el modelo correspondiente de BD para almacenar los datos.
- **“/procesar_respuesta”:** Procesa una respuesta, en caso de obtener todos los datos correctamente se delegan en el modelo correspondiente de BD para almacenar los datos.

- **“/procesar_adivinar”:** Procesa una respuesta, en caso de obtener todos los datos correctamente se delegan en el modelo correspondiente de BD para almacenar los datos.
- **“/amigos”:** Muestra una lista de amigos. También muestra una barra para buscar y añadir nuevos amigos.
- **“/aceptar/:emailAmigo”:** Acepta la petición de amistad.
- **“/rechazar/:emailAmigo”:** Rechaza una petición de amistad.
- **“/amigo/:email”:** Muestra el perfil de un amigo.
- **“/buscar”:** Permite buscar un amigo en la aplicación.
- **“/solicitar_amistad/:id”:** envía una solicitud de amistad al amigo pasado por parámetro.

4. Implementación de la gestión para un usuario logueado

Para saber si un usuario está logueado, guardamos su email, que es el identificador en la tabla usuarios, en un parámetro en la variable session llamado currentUser, el cual guardamos en una variable llamada userEmail en locals a través del middleware CurrentUser. Esto mismo también lo hacemos con los puntos, ya que es información que vamos a necesitar todo el tiempo.
Cuando se loguea un usuario, este se guarda en session, después pasar por el middleware currentUser guardándose en locals. De esta forma, mientras exista la variable session currentUser, la cual se comprueba con el middleware, el usuario está logueado. Al hacer logout, se destruye esta variable session, dejando de estar logueado el usuario.

5. Restricción de acceso a las rutas para usuarios no logueados

A la hora de restringir las rutas para usuarios no logueados, usamos el middleware currentUser, en el cual si no existe la variable en session de currentUser, te redirige a la página de login. Este middleware se usa en todas las funciones donde el usuario necesite estar logueado.

6. Gestión de Errores

Para la gestión de errores disponemos de 2 middleware. El primer middleware es llamado cuando una función llama a su función next pasándole un argumento, el control pasa directamente a nuestro primer manejador de errores disponibles.

A continuación llamamos a otro middleware que es el que gestiona qué tipo de página de error mostrar.
A continuación mostramos cómo se lleva a cabo esta gestión.

```
87   // Captura error 404.
88   app.use(function(req, res, next) {
89       var error = new Error("Not Found");
90       error.status = 404;
91       next(error);
92   });
93
94   // Error Handler.
95   app.use(function(error, request, response, ne
96       if (error.status == 404) {
97           response.render("error404", {
98               url: request.url
99           });
100      } else {
101          response.render("error500", {
102              mensaje: error.message,
103              pila: error.stack
104          });
```