Presentación grupo 05 Aplicación- YOLO

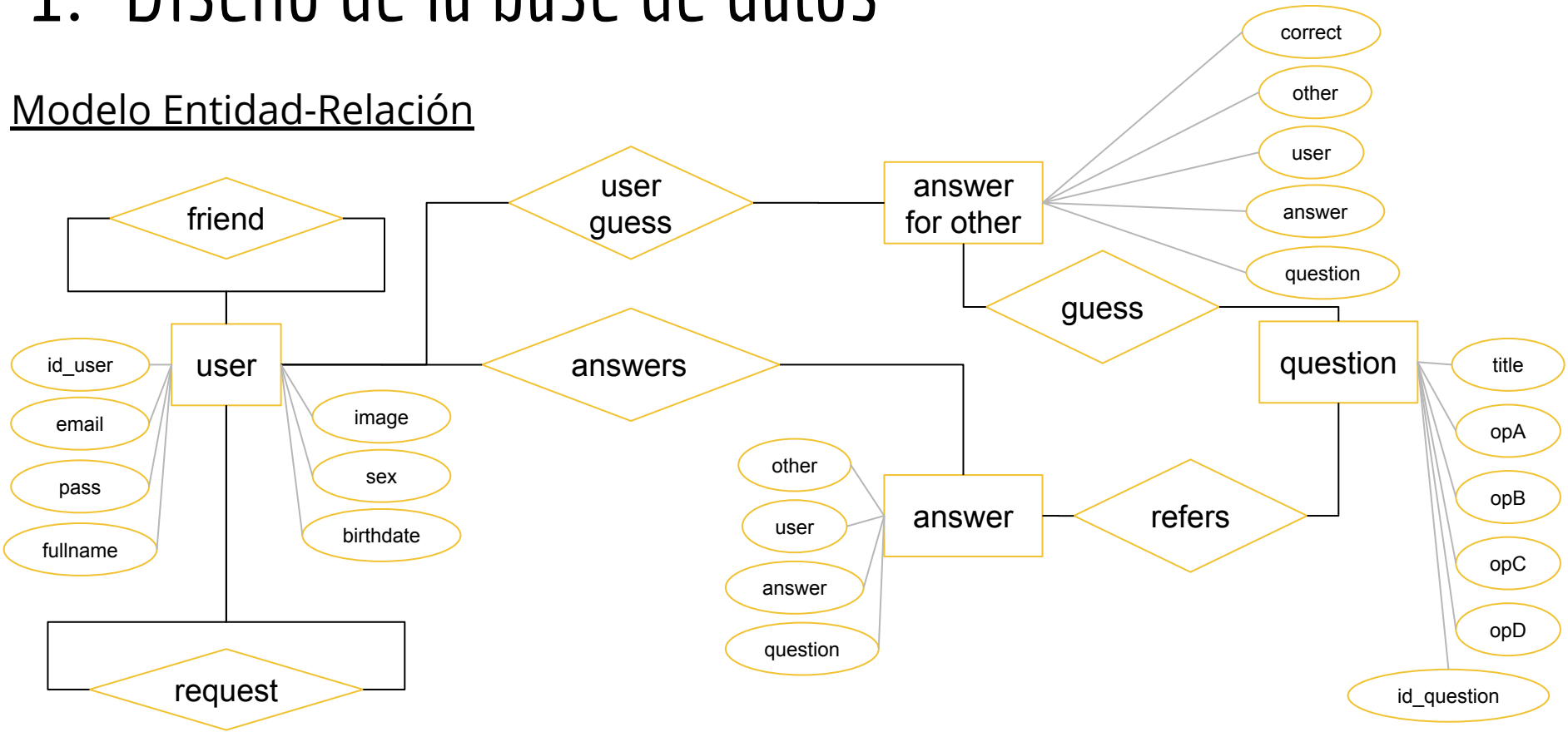
Pablo Agudo Brun
Daniel Fidalgo Panera

Contenido

1. Diseño de la base de datos
2. Estructura de la aplicación
3. Listado de las rutas gestionadas por el servidor
4. Implementación de la sesión para un usuario logueado
5. Restricción de acceso a las rutas para los usuarios no logueados
6. Gestión de los errores 404 y 500

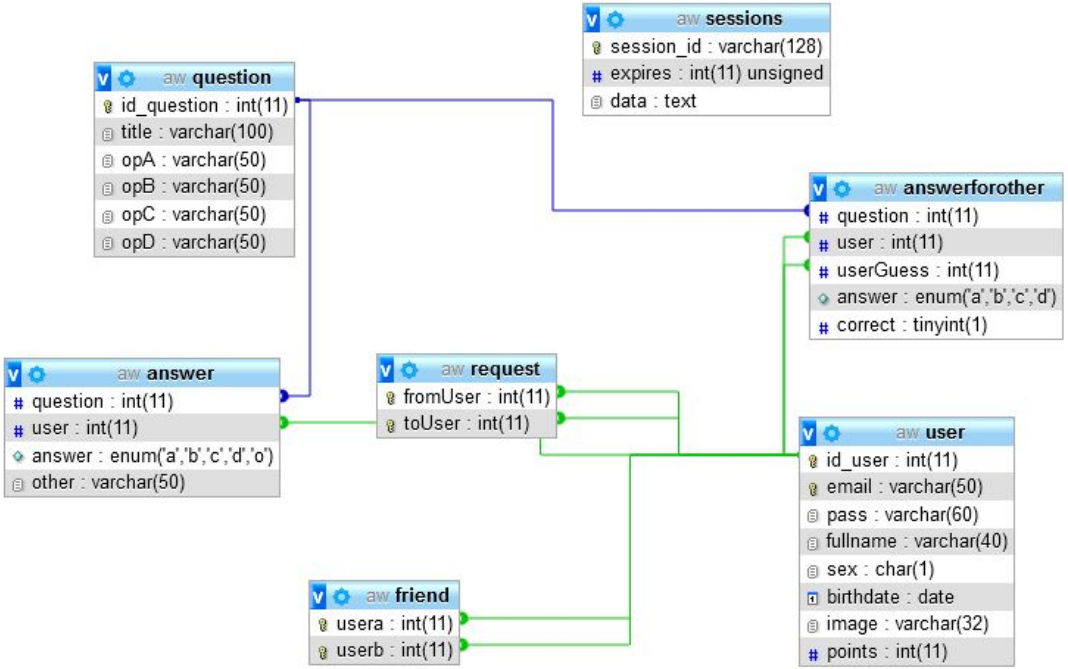
1. Diseño de la base de datos

Modelo Entidad-Relación

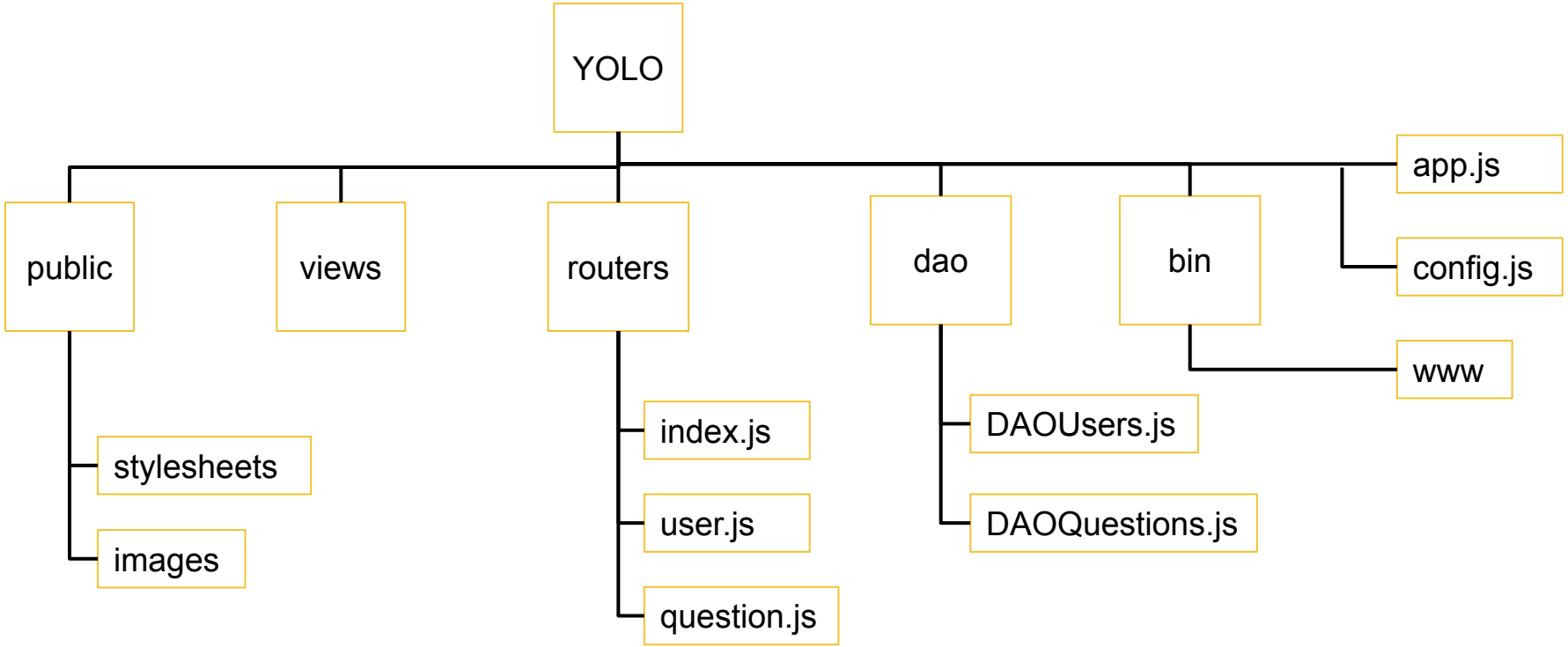


1. Diseño de la base de datos

Modelo relacional



2. Estructura de la aplicación



2. Estructura de la aplicación

Otros módulos

- **bcryptjs**: para la encriptación de la contraseña en la base de datos y evitar guardarla en texto plano.
- **body-parser**: para analizar el contenido de los formularios.
- **cookie-parser**: para gestionar las cookies.
- **ejs**: para poder usar plantillas en documentos HTML con marcadores especiales.
- **express**: para la creación de la aplicación web.
- **express-mysql-session**: para almacenar la información de sesión en la base de datos.
- **express-session**: para usar el atributo session, el cual contiene los datos de sesión de un determinado usuario.
- **express-validator**: para comprobar la validez de los datos introducidos por un usuario en los formularios.
- **http-errors**: para gestionar los códigos de errores en la aplicación.
- **morgan**: para escribir por pantalla peticiones recibidas.
- **multer**: para analizar el contenido de los formularios con codificación multipart/form-data.
- **mysql**: para la conexión con la base de datos relacional.
- **path**: para usar funciones de utilidad para trabajar con nombres de ficheros de manera independiente del SO.

3. Listado de las rutas gestionadas por el servidor

index.js

- ***"/"***
  - get: redirecciona a la página de inicio de la aplicación.

3. Listado de las rutas gestionadas por el servidor

user.js *"/user/"*

- **"/"** :
  - get: redirecciona a /user/login.
- **login**
  - get: renderiza la vista de login.
  - post: redirecciona a /user/profile una vez identificado correctamente al usuario.
- **register**
  - get: renderiza la vista de register.
  - post: redirecciona a /user/login en el caso de validar los campos del formulario y crear al usuario correctamente.
- **modify**
  - get: renderiza la vista de modify.
  - post: redirecciona a la vista de profile en caso de haber modificado los datos de usuario correctamente.
- **profile**
  - get: renderiza la vista de profile con los datos del usuario registrado.
- **uploads/:id**
  - get: envía la imagen de perfil correspondiente del usuario identificada por el parámetro id.

3. Listado de las rutas gestionadas por el servidor

user.js *"/user/"*

- **profile/:id**
  - get: renderiza la vista de profile con los datos del usuario correspondiente al parámetro id.
- **logout**
  - get: elimina los datos de la sesión, limpia las cookies y redirecciona a la vista de login.
- **friends**
  - get: renderiza la vista de friends una vez consultadas las solicitudes de amistad y los amigos del usuario registrado.
- **friends/accept/:id**
  - get: redirecciona a la vista de friends una vez aceptada la solicitud de amistad de un usuario identificado por el parámetro id.
- **friends/reject/:id**
  - get: redirecciona a la vista de friends una vez denegada la solicitud de amistad de un usuario identificado por el parámetro id.
- **search**
  - get: renderiza la vista de search_users una vez buscado a los usuarios que contengan la cadena buscada.
- **request_friend**
  - post: redirecciona a la vista de search una vez enviada la solicitud de amistad a un determinado usuario.
- **profile/image**
  - redirecciona a la vista de profile una vez subida la imagen a la galería del usuario.

3. Listado de las rutas gestionadas por el servidor

question.js *"/question/"*

- **questions**
  - get: renderiza la vista de questions una vez generadas las preguntas aleatorias a mostrar.
- **question/:id**
  - get: renderiza la vista de question de una determinada pregunta identificada por el parámetro id.
- **create_question**
  - get: renderiza la vista de create_question.
- **add_question**
  - post: redirecciona a la vista questions una vez creada la pregunta correctamente.
- **answer/:id**
  - get: renderiza la vista de answer identificada por el parámetro id.
  - post: redirecciona a la vista de question una vez contestada la pregunta.
- **guess/:question/:user**
  - get: renderiza la vista de answerOther la pregunta identificada por el parámetro question y el usuario user
  - post: redirecciona la vista de question una vez respondida la pregunta por otro usuario.

4. Implementación de la sesión para usuario logueado

```
//middleware que contiene los datos de sesion correspondientes al cliente actual
const middlewareSession = session({
        saveUninitialized: false, //indica que no se cree ninguna sesión para los clientes que no estén en la BD de sesiones
        secret: "Xenial Xerus",   //cadena que se utiliza para firmar el SID que se envía al cliente.
        resave: false,            //fuerza a que se guarde, o no, el contenido en la sesión en la BD
        store: sessionStore
});
```

- Utilizamos el middleware express-session para añadir el atributo session al objeto request que contiene los datos de sesión correspondientes al cliente que se está atendiendo.
- Utilizamos también express-mysql-session para almacenar la información de la sesión el la base de datos.

5.1. Restricción de acceso a rutas para usuarios no logueados

```
const redirectLogin = function(req, res, next) {
        if (!req.session.currentUser) {
                res.redirect("/user/login");
        } else {
                next();
        }
}
```

- Comprobamos si el usuario está logueado con req.session.currentUser y en el caso de no estarlo, se redirige a la página de login.
- Se utiliza en las rutas que deberían de ser inaccesibles por un usuario que no esté logueado, por ejemplo, la vista de perfil.

5.2. Restricción de acceso a rutas para usuarios logueados

```
const redirectProfile = function(req, res, next) {
        if (req.session.currentUser) {
                res.redirect("/user/profile");
        } else {
                next();
        }
}
```

- A través de un middleware, comprobamos si el usuario está logueado con req.session.currentUser y en el caso de estarlo, se redirige al perfil del usuario.
- Se utiliza en las rutas que deberían de ser inaccesibles por un usuario logueado, por ejemplo, la vista de login.

6. Gestión de errores

error 404

```
app.use(function(req, res, next) {
        next(createError(404));
});
```

- En el penúltimo lugar la cadena de middlewares hemos colocado este, el cual gestiona los casos en los que una url no haya sido capturada por ningún manejador anterior.
- Este middleware utiliza el módulo http-errors para crear un error 404 que procesaría el siguiente middleware de error.

6. Gestión de errores

error 500

```
app.use(function(err, req, res, next) {
        // set locals, only providing error in development
        res.locals.message = err.message;
        res.locals.error = req.app.get('env') === 'development' ? err : {};

        // render the error page
        res.status(err.status || 500);
        res.render('error');
});
```

- Al final de la cadena de middlewares hemos colocado este middleware, el cual muestra un error recibido si existe y si no, muestra el error 500.