eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Proyecto Skills - Back-end

Jaume Boneu Vidal, Igor Javier Da Silva López,
Haritz Gómez Sarasola y Iker López Dambolenea

Sistemas Web

1 Introducción

Este proyecto consiste en implementar una plataforma de aprendizaje gamificado que fomente el aprendizaje práctico y colaborativo de un tema en concreto.

Este informe explica la parte back-end del proyecto.

2 Estructura del Proyecto

En la parte de back-end hemos creado y añadido nuevas carpetas con el objetivo de completar la estructura que se pide.

La carpeta *routes* contiene los archivos *index.js* y *users.js*. Estos archivos se encargan de definir las rutas de la API, que gestionan las solicitudes entrantes y las redirigen a los controladores correspondientes. *index.js* maneja las rutas generales de la aplicación, mientras que *users.js* se especializa en las operaciones relacionadas con los usuarios, como la creación de usuarios o la modificación de las contraseñas.

En el directorio *models*, se encuentran los archivos *badgeModel.js*, *evidencesModel.js*, *skillModel.js* y *userModel.js*. Estos archivos definen las estructuras de los datos que se manipulan en la base de datos.

Finalmente, el directorio *views* contiene las distintas vistas que permiten gestionar la interacción entre el usuario y la aplicación. Más adelante explicaremos las principales vistas.

3 Modelos

La base de datos ha sido implementada mediante Mongoose, una librería de MongoDB. Se ha definido un modelo para los usuarios, skills, badges y evidencias. A continuación se muestra cada uno de ellos:

Para cada usuario se ha definido un nombre de usuario, contraseña, score, si es o no admin y las skills completadas.

```
const userSchema = new mongoose.Schema({
    username : {
        type: String,
        required: true,
        minlength: 3,
        maxLength: 20,
        unique: true
    },
    password: {
        type: String,
        required: true,
        minlength: 6,
    },
    score: {
        type: Number,
        required: true,
    },
    admin: {
        type: Boolean,
        required: true,
    },
    completedSkills: {
        type: Array,
        required: true,

    }
})
```

Para cada skill se ha definido un id, texto, icono, set, tasks, recursos, descripción y score.

```
const skillScheme = new mongoose.Schema({
    identifier : {
        type: Number,
        required: true,
        unique: true
    },
    text: {
        type: String,
        required: true,
        minlength: 6,
    },
    icon: {
        type: String,
        required: false,
    },
    set: {
        type: String,
        required: true,
    },
    tasks: {
        type: Array,
        required: true,
        minlength: 1,
    },
    resources: {
        type: Array,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    score: {
        type: Number,
        required: true,
        default: 1
    }
})
```

Para cada badge, se ha definido el rango, número mínimo y máximo de bitpoints y ruta png del badge.

```
const badgeSchema = new mongoose.Schema({
    rango: { type: String, required: true },
    bitpoints_min: { type: Number, required: true },
    bitpoints_max: { type: Number, required: true },
    png: { type: String, required: true }
});
```

Para cada evidencia, se ha definido el id de la skill, nombre de usuario, url de la evidencia y si ha sido aprovada o no.

```
const evidenceSchema = new Schema({
    skillId: { type: String, required: true },
    username: { type: String, required: true },
    url: { type: String, required: true },
    approved: { type: Boolean, default: false }
});
```

4 Views

Hemos definido varias vistas. Las principales son las siguientes:

- *Login*: Esta vista permite a los usuarios iniciar sesión. Si no tienen una cuenta, se pueden registrar.
- *Register*: Esta vista permite a los usuarios registrarse en el sistema.
- *Manage*: Esta vista permite al administrador ver los usuarios registrados en el sistema y cambiar su contraseña.
- *Add Skill*: Esta vista permite añadir una nueva skill al sistema.
- *Edit Skill*: Esta vista permite editar los datos de una skill ya existente.
- *Add Skill*: Esta vista permite añadir una nueva skill al sistema.
- *Skill Specs*: Esta vista permite ver las especificaciones de una skill, completar tareas y enviar evidencias.
- *Admin Badges*: Esta vista permite administrar los badges del sistema y editar o eliminar medallas.

5 Servidor

En el servidor, el manejo de las peticiones HTTP se realiza a través de endpoints y middlewares definidos en los archivos de la carpeta routes.

Los endpoints son los puntos de interacción directa entre el cliente y el servidor. Cada endpoint representa una funcionalidad específica y está vinculado a una ruta y un método HTTP. La idea clave es que cada endpoint debe:

1. Procesar solicitudes específicas: Manejar la lógica necesaria para responder al cliente según el tipo de acción requerida (registro, login, envío de datos, etc.).

2. Separación/división de solicitudes en la lógica de negocio: Enfocarse únicamente en lo que esa ruta necesita hacer (validar datos, interactuar con la base de datos, etc.), dejando las tareas generales a los middlewares.

Por ejemplo:

- En un endpoint para login, se valida el usuario, se crea la sesión y se responde con una redirección o un mensaje de error.
- En otro para manejar usuarios, se realizan acciones administrativas, como cambiar contraseñas o editar información.

Los middlewares son funciones intermedias que se ejecutan antes de que el servidor llegue a un endpoint. Su propósito principal es manejar tareas comunes o transversales para todas (o muchas) rutas, como:

1. Gestión de sesiones: Verificar si un usuario está autenticado o inicializar su sesión.
2. Validaciones globales: Comprobar que los datos recibidos son válidos antes de pasar al endpoint.
3. Configuraciones compartidas: Preparar datos para que los endpoints puedan usarlos sin redundancia.

La idea detrás de los middlewares es:

- Evitar duplicación de código: Reutilizar lógica común, como autenticaciones, en múltiples endpoints.
- Mantener el flujo controlado: Detener la ejecución si algo falla (por ejemplo, rechazar una solicitud no autenticada antes de que llegue al endpoint).
- Facilitar el mantenimiento: Al concentrar lógica general en un lugar, cualquier cambio se hace una sola vez.

Los middlewares preparan el camino para que los endpoints funcionen correctamente. Mientras que los endpoints responden a solicitudes específicas, los middlewares se aseguran de que todo esté en orden antes de llegar a ellos. Esto divide responsabilidades y hace que el código sea más limpio, eficiente y fácil de escalar.