IES Virgen del Carmen

JUNTA DE ANDALUCIA
CONSEJERÍA DE EDUCACIÓN

Departamento de Informática

Adivimat Accesible

Carlos Daniel Ondo Angue

Junio 2024

UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Índice

1 Introducción

1.1 Datos del proyecto

Nombre	Apellidos	Título	Ciclo	Año	Centro Educativo
Carlos Daniel	Ondo Angue	Adivimat Accesible	Técnico Superior en Desarrollo de Aplicaciones Multiplataforma	2024	I.E.S. VIRGEN DEL CARMEN

1.2 Planificación

1.3 Semana del 15 al 21 de abril

- **Organización y fases del trabajo:** Planificación, recursos que podría necesitar, viabilidad. Tecnologías que queremos usar.
- **Ideas iniciales sobre el mismo:** los “deseos” o “características” de SCRUM.
- **Elementos del proyecto:** qué pretendemos generar: un manual o tutorial, un producto software, la memoria y la presentación.
- **Preparación de la plantilla del documento:** a entregar, o, en su defecto el repositorio vacío para empezar a documentar en Markdown.
- **Investigación de estudios y proyectos similares:** plasmarlo en la documentación.

1.4 Semana del 22 al 28 de abril

- **Introducción:** generar este apartado en la documentación.
- **Objetivos definitivos:** exactamente qué estamos haciendo.
- **Material y recursos a utilizar:** recoger detalladamente todos los recursos que se disponen y/o necesitarán en la documentación.
- **Métodos seguidos en el proceso, metodologías, tecnologías:** ej. por qué usar un lenguaje o framework concreto y no otro.

1.5 Semana del 29 de abril al 5 de mayo

- **SPRINT 1:** Ya hay que tener un producto funcional, aunque sea un esqueleto y haga muy poco.

- **Resultados iniciales:** primeros “bocetos” del programa.
- **Análisis de cambios necesarios:** en requisitos o tecnologías inicialmente planificadas y explicar si hay algún cambio por qué se ha hecho. Esto se plasma en la documentación.

1.6 Semana del 6 al 12 de mayo

- **SPRINT 2:** Resultados intermedios (demo funcional).
- **Primera revisión de la documentación:** para ver que estén todos los puntos necesarios.

1.7 Semana del 20 al 26 de mayo

- **SPRINT 3:** Aplicación casi terminada, para corregir bugs, problemas de interfaz…
- **Segunda revisión del documento:** donde ya estén todos los apartados necesarios.
- **Preparación de la presentación.**

1.8 Semana del 27 de mayo al 2 de junio

- **SPRINT 4:** Resultados finales (proyecto terminado: tutorial, aplicación…).
- **Entrega del documento final.**

1.9 Semana del 3 al 9 de junio

- **Organización de la presentación.**
- **Entrega de la presentación para la exposición.**

1.10 Semana del 10 al 17 de junio

- **Presentación de proyectos.**

2 Tecnologías

2.1 Back-end:

2.1.1 1. Node.js

- **Qué es**: Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.
- **Por qué lo uso**: Node.js permite construir aplicaciones de red escalables y rápidas gracias a su modelo de E/S sin bloqueo y basado en eventos. Es ideal para aplicaciones en tiempo real y APIs RESTful.

2.1.2 2. Express.js

- **Qué es**: Express es un framework para aplicaciones web en Node.js.
- **Por qué lo uso**: Simplifica la creación y el manejo de rutas y middleware, permitiendo una configuración rápida y eficiente de la aplicación web y sus rutas.

2.1.3 3. MongoDB

- **Qué es**: MongoDB es una base de datos NoSQL orientada a documentos.
- **Por qué lo uso**: MongoDB proporciona una gran flexibilidad y escalabilidad, permitiendo almacenar datos en documentos JSON dinámicos. Es fácil de integrar con Node.js a través del driver de MongoDB para Node.js.

2.1.4 4. Mongoose

- **Qué es**: Mongoose es una biblioteca de Node.js que proporciona una solución basada en esquemas para modelar datos en MongoDB.
- **Por qué lo uso**: Simplifica las interacciones con MongoDB proporcionando validación de esquemas, casting de tipos y otras funcionalidades útiles.

2.1.5 5. JWT (JSON Web Tokens)

- **Qué es**: JWT es un estándar abierto para crear tokens de acceso que permiten la verificación y autorización de usuarios.

- **Por qué lo uso**: JWT es ideal para manejar la autenticación de usuarios en aplicaciones web y móviles, proporcionando una manera segura y compacta de transmitir información entre el cliente y el servidor.

2.1.6 6. bcrypt

- **Qué es**: bcrypt es una librería para el hashing de contraseñas.
- **Por qué lo uso**: Garantiza que las contraseñas de los usuarios estén almacenadas de manera segura mediante el uso de un algoritmo de hash que incluye un salt para mayor seguridad.

2.1.7 7. dotenv

- **Qué es**: dotenv es una librería que carga variables de entorno desde un archivo `.env` al `process.env` de Node.js.
- **Por qué lo uso**: Permite manejar de manera segura y sencilla las configuraciones sensibles y las credenciales de la aplicación.

2.1.8 8. CORS (Cross-Origin Resource Sharing)

- **Qué es**: CORS es un mecanismo que permite solicitudes HTTP desde diferentes dominios.
- **Por qué lo uso**: Necesito CORS para permitir que el frontend de la aplicación pueda comunicarse con el backend alojado en un dominio diferente, para mejoras en el futuro.

2.1.9 9. Docker

- **Qué es**: Docker es una plataforma para desarrollar, enviar y ejecutar aplicaciones dentro de contenedores.
- **Por qué lo uso**: Docker permite empaquetar la aplicación y sus dependencias en un contenedor, asegurando que funcione en cualquier entorno. Esto facilita el despliegue y la escalabilidad de la aplicación.

2.2 Estructura

2.2.1 1. `app.js`

- Configuración del servidor, conexión a la base de datos y establecimiento de las rutas principales de la aplicación.

- **Tecnologías usadas**: Express.js, Mongoose, dotenv, CORS.

2.2.2 2. Rutas

- **`authRoutes.js`**: Maneja las rutas de autenticación (`/register`, `/login`, `/logout`) y la verificación de administradores.
- **`temasRoutes.js`**: Maneja las rutas CRUD para `temas`, `subtemas` y `adivinanzas`.
- **Tecnologías usadas**: Express.js.

2.2.3 3. Controladores

- **`authController.js`**: Controla la lógica para el registro, inicio de sesión y verificación de usuarios.
- **`temaController.js`**: Controla la lógica CRUD para `temas`, `subtemas` y `adivinanzas`.
- **Tecnologías usadas**: Mongoose, bcrypt, JWT.

2.2.4 4. Modelos

- **`user.js`**: Define el esquema y modelo de `User`.
- **`tema.js`**: Define el esquema y modelo de `Tema`.
- **`subtema.js`**: Define el esquema y modelo de `Subtema`.
- **`adivinanza.js`**: Define el esquema y modelo de `Adivinanza`.
- **Tecnologías usadas**: Mongoose.

2.2.5 5. Utilidades

- **`generateToken.js`**: Función para generar tokens JWT.
- **Tecnologías usadas**: JWT.

2.2.6 6. Carpeta Stack con el docker-compose

- **Expicado aqí**

2.3 Front-end:

2.3.1 React.js

- **Qué es**: React.js es una biblioteca de JavaScript para construir interfaces de usuario.
- **Por qué lo uso**: React.js permite la creación de aplicaciones web dinámicas y rápidas gracias a su enfoque basado en componentes y su eficiente manejo del DOM virtual.

2.3.2 Redux

- **Qué es**: Redux es una biblioteca para el manejo del estado de la aplicación.
- **Por qué lo uso**: Facilita la gestión del estado en aplicaciones React, permitiendo un flujo de datos predecible y simplificado a través de un almacén centralizado.

2.3.3 React Router

- **Qué es**: React Router es una biblioteca para manejar la navegación y las rutas en aplicaciones React.
- **Por qué lo uso**: Permite definir rutas y gestionar la navegación de manera sencilla y eficiente, mejorando la experiencia de usuario en aplicaciones de una sola página (SPA).

2.3.4 Axios

- **Qué es**: Axios es una biblioteca para hacer solicitudes HTTP desde el navegador.
- **Por qué lo uso**: Facilita la comunicación con el backend, permitiendo realizar peticiones API de manera sencilla y con soporte para promesas.

2.3.5 aria-live

- **Qué es**: `aria-live` es un atributo de accesibilidad utilizado para anunciar dinámicamente las actualizaciones de contenido a los usuarios de tecnologías asistivas, como lectores de pantalla.
- **Por qué lo uso**: Utilizo `aria-live` para asegurar que los mensajes importantes y las actualizaciones en la aplicación sean accesibles para todos los usuarios, incluyendo aquellos con discapacidades visuales. Esto mejora la usabilidad y accesibilidad de la aplicación, garantizando que todos los usuarios reciban la información crítica de manera oportuna.

2.4 Estructura

2.4.1 1. App.js

- Configuración principal de la aplicación, incluyendo rutas y componentes básicos.
- **Tecnologías usadas**: React.js, React Router.

2.4.2 2. Rutas

- **About.js**: Página de información sobre la aplicación.
- **AddRiddle.js**: Página para añadir nuevas adivinanzas.
- **AddSubtheme.js**: Página para añadir nuevos subtemas.
- **AddTheme.js**: Página para añadir nuevos temas.
- **EditTheme.js**: Página para editar temas existentes.
- **EditSubtheme.js**: Página para editar subtemas existentes.
- **EditRiddle.js**: Página para editar adivinanzas existentes.
- **Home.js**: Página principal de la aplicación.
- **Login.js**: Página de inicio de sesión.
- **Register.js**: Página de registro de usuarios administradores.
- **Gestion.js**: Página para la gestión de temas, subtemas y adivinanzas.
- **Tecnologías usadas**: React.js, React Router.

2.4.3 3. Controladores de Estado

- **AuthProvider.js**: Controla la lógica de autenticación y el estado del usuario.
- **ThemeContext.js**: Controla la lógica de temas y subtemas.
- **Tecnologías usadas**: React.js, Redux.

2.4.4 4. Configuración

- **axiosConfig.js**: Configuración de Axios para manejar solicitudes HTTP.
- **validation.js**: Funciones de validación para entradas de usuario.
- **Tecnologías usadas**: Axios, JavaScript.

2.4.5 5. Componentes de Interfaz

- **Footer.js**: Componente para el pie de página de la aplicación.

- **MenuAppBar.js**: Barra de navegación superior.
- **LogoutButton.js**: Botón para cerrar sesión.
- **PrivateRoute.js**: Componente para proteger rutas privadas.
- **RiddleSelector.js**: Selector de adivinanzas.
- **RiddleTable.js**: Tabla de adivinanzas.
- **ThemeList.js**: Lista de temas.
- **SubthemeList.js**: Lista de subtemas.
- **ThemeManager.js**: Gestión de temas y subtemas.
- **Tecnologías usadas**: React.js, CSS.

2.4.6 6. Estilos

- **App.css**: Define los estilos globales de la aplicación.
- **Footer.css**: Define los estilos específicos para el pie de página.
- **Tecnologías usadas**: CSS.

2.4.7 7. Gestión de Contenidos

- **ThemeManager.js**: Componente principal para la gestión de temas, subtemas y adivinanzas.
  - **Qué es**: Proporciona la interfaz para gestionar los temas, subtemas y adivinanzas.
  - **Por qué lo uso**: Permite a los administradores añadir, editar y eliminar contenido de manera eficiente.

2.4.8 8. Autenticación y Autorización

- **Login.js**: Página de inicio de sesión.
  - **Qué es**: Permite a los usuarios autenticarse en la aplicación.
  - **Por qué lo uso**: Controla el acceso a las funcionalidades protegidas de la aplicación.
- **Register.js**: Página de registro de usuarios.
  - **Qué es**: Permite a los nuevos usuarios administradores registrarse en la aplicación.
  - **Por qué lo uso**: Facilita la incorporación de nuevos admin a la aplicación.
- **LogoutButton.js**: Botón para cerrar sesión.
  - **Qué es**: Permite a los usuarios cerrar su sesión de manera segura.
  - **Por qué lo uso**: Garantiza que los usuarios puedan cerrar sesión y proteger su información.

- `PrivateRoute.js`: Componente para proteger rutas privadas.
  - **Qué es**: Protege rutas que solo deben ser accesibles para usuarios autenticados.
  - **Por qué lo uso**: Asegura que solo los usuarios autenticados puedan acceder a ciertas funcionalidades.

2.4.9 Despliegue del frontend

React

3 Instalación y configuración de MongoDB

Para crear la infraestructura de servicios necesaria, en vez de instalar MongoDB, vamos a usar un contenedor para el servidor MongoDB y otro contenedor para Mongo-Express (una interfaz para interactuar con la base de datos).

Creamos la carpeta `stack` y dentro de ésta el archivo `docker-compose.yml` con el siguiente contenido:

```yaml
# Use root/example as user/password credentials
version: '3.1'

services:

  mongo:
    image: mongo
    restart: 'no'
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 83uddjfp0cmMD
    ports:
      - 27017:27017

  mongo-express:
    image: mongo-express
    restart: 'no'
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_BASICAUTH_USERNAME: mongo
      ME_CONFIG_BASICAUTH_PASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: 83uddjfp0cmMD
      ME_CONFIG_MONGODB_URL: mongodb://root:83uddjfp0cmMD@mongo:27017/
```

```yaml
version: '3.1'
```

Esta línea especifica la versión de la sintaxis de `docker-compose` que se está utilizando. En este caso, es la versión 3.1.

```yaml
services:
  mongo:
    image: mongo
    restart: 'no'
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 83uddjfp0cmMD
    ports:
      - 27017:27017
```

Aquí se define un servicio llamado `mongo`. Este servicio utiliza la imagen oficial de MongoDB (`mongo`). Algunas configuraciones clave incluyen:

- `restart: 'no'`: Esto indica que el contenedor no se reiniciará automáticamente a menos que se haga manualmente.
- `environment`: Establece variables de entorno necesarias para configurar la base de datos MongoDB. En este caso, se especifica el nombre de usuario (usuario) y la contraseña (contraseña) para el usuario root de MongoDB.
- `ports`: Mapea el puerto 27017 del host al puerto 27017 del contenedor, lo que permite la comunicación con la instancia de MongoDB.

```
  mongo-express:
    image: mongo-express
    restart: 'no'
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_BASICAUTH_USERNAME: mongo
      ME_CONFIG_BASICAUTH_PASSWORD: contraseña
      ME_CONFIG_MONGODB_ADMINUSERNAME: usuario
      ME_CONFIG_MONGODB_ADMINPASSWORD: contraseña
      ME_CONFIG_MONGODB_URL: mongodb://usuario:contraseña@mongo:27017/
```

Aquí se define un segundo servicio llamado `mongo-express`. Este servicio utiliza la imagen de Mongo Express (`mongo-express`). Algunas configuraciones clave son similares al servicio `mongo`, pero específicas de Mongo Express:

- `restart: 'no'`: Al igual que en el servicio `mongo`, indica que el contenedor no se reiniciará automáticamente.
- `ports`: Mapea el puerto 8081 del host al puerto 8081 del contenedor, permitiendo acceder a la interfaz web de Mongo Express.
- `environment`: Configura las variables de entorno necesarias para la autenticación en MongoDB.
- `ME_CONFIG_MONGODB_URL`: Especifica la URL de conexión a la base de datos MongoDB. En este caso, utiliza el usuario root y la contraseña proporcionados en las variables de entorno anteriores.
- `ME_CONFIG_BASICAUTH_USERNAME` y `ME_CONFIG_BASICAUTH_PASSWORD` definen el usuario y contraseña para la interfaz Web.

Para poner en marcha estos servicios, debes tener Docker y Docker Compose instalados. Luego, ejecuta el siguiente comando en el directorio donde se encuentra el archivo `docker-compose.yml`:

```
docker-compose up -d
```

Esto descargará las imágenes necesarias, creará y ejecutará los contenedores según la configuración proporcionada. Después de que los contenedores estén en funcionamiento, podrás acceder a MongoDB a través del puerto 27017 y a Mongo Express a través del puerto 8081 en tu máquina local.

4 React

4.1 Requisitos Previos

Para trabajar en el proyecto en React, necesitamos el siguiente software:

1. **Node.js**: Necesario para el proceso de transpilación, empaquetado, lint, etc. Puedes descargarlo e instalarlo desde nodejs.org.
2. **npm (Node Package Manager)**: Viene incluido con la instalación de Node.js y se usa para gestionar las dependencias del proyecto.

4.2 Instalación del Proyecto

Sigue estos pasos para configurar el entorno de desarrollo y ejecutar el proyecto React.

4.2.1 Paso 1: Clonar el Repositorio

```
git clone <https://github.com/cdoaweb/pci-adivimat.git>
cd pci-adivimat/frontend/adivimat-accesible
```

4.2.2 Paso 2: Instalar Dependencias

Instala las dependencias necesarias ejecutando el siguiente comando en el directorio del proyecto. Las dependencias necesarias están listadas en el archivo ‘package.json.

```
npm install
```

4.2.3 Dependencias en `package.json`

Estas son las dependencias que se instalarán:

```
"dependencies": {
  "@emotion/react": "^11.11.4",
  "@emotion/styled": "^11.11.5",
  "@mui/material": "^5.15.18",
  "@testing-library/jest-dom": "^5.17.0",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^1.6.8",
  "dotenv": "^16.4.5",
```

```
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-router-dom": "^6.23.1",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
}
```

4.2.4 Paso 3: Ejecutar la Aplicación

Para iniciar la aplicación en modo desarrollo, usa el siguiente comando:

```
npm start
```

Esto abrirá una nueva pestaña en el navegador con la aplicación React en funcionamiento.

5 Implementación

5.1 Back-end:

5.1.1 Conexión a la Base de Datos

```
// app.js
const mongoose = require('mongoose');
const dotenv = require('dotenv').config();

const mongoUri = process.env.MONGO_URI;
mongoose.connect(mongoUri, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('Conexión exitosa a MongoDB'))
.catch(err => console.error('Error al conectar con MongoDB:', err));
```

5.1.2 Registro de Usuario admin para gestionar adivinanzas

```
// authController.js
const bcrypt = require('bcrypt');
const User = require('../models/user');
const generateToken = require('../utils/generateToken');

exports.register = async (req, res) => {
  try {
    const { username, email, password, adminCode } = req.body;

    // Verificar si el código de admin es correcto para permitir la
       creación de un admin
    if (adminCode !== process.env.ADMIN_CODE) {
      return res.status(401).json({ message: 'Unauthorized to create an
          admin' });
    }

    // Encriptar la contraseña antes de guardar el usuario
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({ username, email, password: hashedPassword,
       isAdmin: true });
    await user.save();

    // Generar el token
    const token = generateToken(user._id, user.isAdmin);

    res.status(201).json({ message: 'Admin created successfully', token
        });
```

```
  } catch (error) {
    res.status(500).json({ message: 'Error registering new admin: ' +
        error.message });
  }
};
```

5.1.3 Generación de Token JWT

```
// generateToken.js
const jwt = require('jsonwebtoken');

// Generar un JWT
const generateToken = (userId, isAdmin) => {
  // Carga la clave secreta desde las variables de entorno
  const secretKey = process.env.JWT_SECRET;

  // Información del token
  const payload = {
    id: userId,
    isAdmin: isAdmin,
  };

  // Opciones del token
  const options = {
    expiresIn: '1h', // El token expira en 1 hora
  };

  // Crear el token
  const token = jwt.sign(payload, secretKey, options);
  return token;
};

module.exports = generateToken;
```

5.1.4 Modelo de Tema con Subtemas y Adivinanzas

```
// tema.js
const mongoose = require('mongoose');
const SubtemaSchema = require('./subtema').schema;

const TemaSchema = new mongoose.Schema({
  tema: String,
  subtemas: [SubtemaSchema]
});

module.exports = mongoose.model('Tema', TemaSchema);

```

```
// subtema.js
const mongoose = require('mongoose');
const AdivinanzaSchema = require('./adivinanza').schema;

const SubtemaSchema = new mongoose.Schema({
  name: String,
  adivinanzas: [AdivinanzaSchema]
});

module.exports = mongoose.model('Subtema', SubtemaSchema);

// adivinanza.js
const mongoose = require('mongoose');

const AdivinanzaSchema = new mongoose.Schema({
  pregunta: {
    type: String,
    trim: false
  },
  respuesta: {
    type: String,
    trim: false
  },
  intentos: {
    type: Number,
    default: 5,
    min: [0, 'Los intentos no pueden ser negativos']
  },
  puntos: {
    type: Number,
    default: 0,
    min: [0, 'Los puntos no pueden ser negativos']
  },
  respuestaRevelada: {
    type: Boolean,
    default: false
  }
});

AdivinanzaSchema.index({ pregunta: 1 });

AdivinanzaSchema.methods.verificarRespuesta = function (
    respuestaUsuario) {
  return this.respuesta.toLowerCase() === respuestaUsuario.toLowerCase
      ();
};

module.exports = mongoose.model('Adivinanza', AdivinanzaSchema);
```

5.1.5 .env

ADMIN_CODE= (El proporcionado) JWT_SECRET=(El proporcionado) MONGO_URI=mongodb://usuario:contraseña@loc
BACKEND_PORT=8000

6 Conclusiones

A lo largo de este proyecto, he integrado un conjunto robusto de tecnologías que, en conjunto, han permitido la creación de una aplicación web eficiente, segura, escalable y accesible. La elección de herramientas específicas para el back-end y el front-end ha sido crucial para lograr estos objetivos.

6.1 Tecnologías Back-end

6.1.1 Node.js y Express.js

Node.js, con su modelo de E/S sin bloqueo, ha sido fundamental para manejar múltiples conexiones simultáneamente, lo que es ideal para aplicaciones en tiempo real. Express.js ha simplificado la configuración del servidor y el manejo de rutas, permitiendo un desarrollo ágil y una estructura clara y mantenible.

6.1.2 MongoDB y Mongoose

La flexibilidad y escalabilidad de MongoDB han sido esenciales para gestionar los datos dinámicos de la aplicación. Mongoose ha proporcionado una capa adicional de comodidad, facilitando la validación y el modelado de datos con esquemas definidos, lo que ha mejorado la integridad y consistencia de la base de datos.

6.1.3 JWT y bcrypt

La implementación de JWT ha asegurado una gestión eficaz y segura de la autenticación de usuarios, permitiendo una comunicación confiable entre el cliente y el servidor. bcrypt ha añadido una capa de seguridad adicional mediante el hashing de contraseñas, protegiendo así los datos sensibles de los usuarios.

6.1.4 Docker

El uso de Docker ha sido decisivo para garantizar que mi aplicación se despliegue de manera uniforme en cualquier entorno, eliminando problemas de configuración y facilitando el escalado de la aplicación.

6.2 Tecnologías Front-end

6.2.1 React.js y Redux

React.js, con su enfoque basado en componentes, ha permitido la creación de una interfaz de usuario dinámica y reactiva. Redux ha simplificado la gestión del estado de la aplicación, asegurando un flujo de datos predecible y facilitando el manejo de estados complejos.

6.2.2 React Router y Axios

React Router ha sido fundamental para la navegación eficiente dentro de la aplicación, mejorando la experiencia del usuario. Axios ha simplificado la comunicación con el back-end, permitiendo realizar peticiones HTTP de manera fluida y manejable.

6.2.3 Accesibilidad con aria-live

La implementación de `aria-live` ha mejorado significativamente la accesibilidad de la aplicación, asegurando que las actualizaciones de contenido sean anunciadas a los usuarios de tecnologías asistivas. Esto ha sido crucial para garantizar que todos los usuarios, independientemente de sus capacidades, puedan utilizar la aplicación de manera efectiva.

6.3 Cosas Nuevas que He Aprendido

A lo largo del desarrollo de este proyecto, he aprendido a:

1. Integrar diferentes tecnologías y herramientas para crear una aplicación completa y funcional.
2. Implementar medidas de seguridad como JWT y bcrypt para proteger la información del usuario.
3. Utilizar Docker para asegurar la consistencia y escalabilidad del entorno de desarrollo y producción.
4. Aplicar principios de accesibilidad web utilizando `aria-live` para mejorar la experiencia de los usuarios en mi misma condición.

6.4 Cosas que Haría de Otra Manera

Con la experiencia adquirida, hay varias cosas que haría de manera diferente en futuros proyectos:

1. **Mejorar la organización del código**: Reestructuraría algunos componentes y módulos para mejorar la mantenibilidad y legibilidad del código.

2. **Automatizar pruebas**: Implementaría un sistema de pruebas automatizadas para garantizar la calidad del código y reducir errores en el futuro.
3. **Mejorar la documentación**: Dedicaría más tiempo a documentar el código y las funcionalidades de la aplicación para facilitar futuras actualizaciones y mantenimiento.

6.5 Qué Haré Después

Mirando hacia el futuro, planeo:

1. **Ampliar las funcionalidades de la aplicación**: Añadir nuevas características basadas en las necesidades y el feedback de los usuarios.
2. **Profundizar en nuevas tecnologías como React Native**: Explorar otras herramientas y frameworks que puedan mejorar aún más mis capacidades de desarrollo.
3. **Mejorar la accesibilidad**: Continuar mejorando la accesibilidad de la aplicación para asegurar que sea inclusiva para todos los usuarios.
4. **Formar una comunidad**: Crear una comunidad alrededor de la aplicación donde los usuarios puedan compartir ideas, sugerencias y colaborar en el desarrollo de nuevas funcionalidades.

6.6 Conclusión Personal

Como finalista de DAM y usuario de tecnologías de asistencia, este proyecto ha representado un desafío y un aprendizaje significativo. He aprendido a integrar tecnologías modernas y a considerar la accesibilidad como un componente central del desarrollo de aplicaciones. Este enfoque no solo mejora la usabilidad para personas con discapacidades, sino que también enriquece la experiencia de todos los usuarios.

Este proyecto demuestra que con las herramientas adecuadas y un enfoque inclusivo, es posible desarrollar aplicaciones web robustas, seguras y accesibles que cumplen con las necesidades actuales del mercado y de la comunidad. Estoy orgulloso del trabajo realizado y confío en que esta aplicación será útil y efectiva para sus usuarios finales.

7 Bibliografía

Curso de MongoDB https://openwebinars.net/academia/aprende/mongodb/

- **Documentación oficial de Mongo**: https://docs.mongo.org.
- **Mongo Query Language**: https://www.mongodb.com/docs/manual/tutorial/query-documents/.

Con este curso he ampliado mis conocimientos de MongoDB, a parte de los adquiridos en clase.

7.1 Curso de React Native para principiantes

https://openwebinars.net/academia/aprende/react-native-principiantes/

https://reactnative.dev/docs/environment-setup

https://blog.logrocket.com/how-to-build-ios-apps-using-react-native/

https://necolas.github.io/react-native-web/docs/accessibility/#:~:text=Accessibility%20in%20React%20Native%20for,attributes%2C%20and%20ARIA%20in%20HTML

Con esta formación he tenido mi primer contacto con la tecnología React Native, con la que en un futuro pretendo migrar el frontend de la aplicación. Cuando se actualice y utilice mejor los componentes de accesibilidad para Android.

7.2 PlantUML:

https://plantuml.com/es/

Con esta herramienta he aprendido a crear diagramas desde VScode, de forma accesible para mí a partir de código y no de forma visual con herramientas como Modelio.

7.3 Tutoriales de Express y MongoDB

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose.

https://www.freecodecamp.org/news/build-a-restful-api-using-node-express-and-mongodb/

documentación de React.

ARIA live.

MUI

8 Diagrama de la base de datos

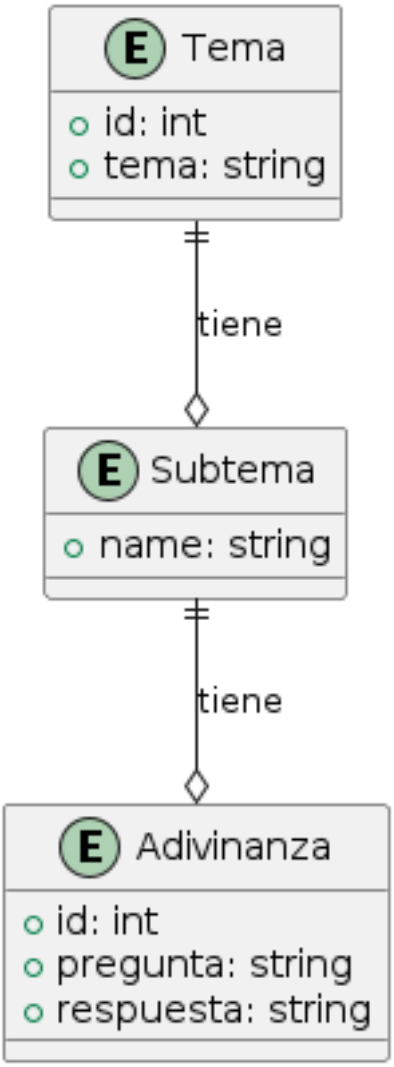

Figura 1: Diagrama Entidad Relación

```
@startuml Diagrama entidad relación

entity Tema {
    +id: int
    +tema: string
}

entity Subtema {
    +name: string
}
```

```
11
12 entity Adivinanza {
13     +id: int
14     +pregunta: string
15     +respuesta: string
16 }
17
18 Tema ||--o Subtema : tiene
19 Subtema ||--o Adivinanza : tiene
20
21 @enduml
```

8.1 Entidades

- **Tema**
- **Subtema**
- **Adivinanza**

8.2 Atributos

- **Tema**: id, tema
- **Subtema**: name
- **Adivinanza**: id, pregunta, respuesta

8.3 Relaciones

- Un **Tema** puede tener múltiples **Subtemas**.
- Un **Subtema** puede tener múltiples **Adivinanzas**.

8.4 Descripción de las relaciones:

- Un **Tema** tiene una relación uno a muchos con **Subtema**.
- Un **Subtema** tiene una relación uno a muchos con **Adivinanza**.

9 Descripción de Diagramas

9.1 Backend

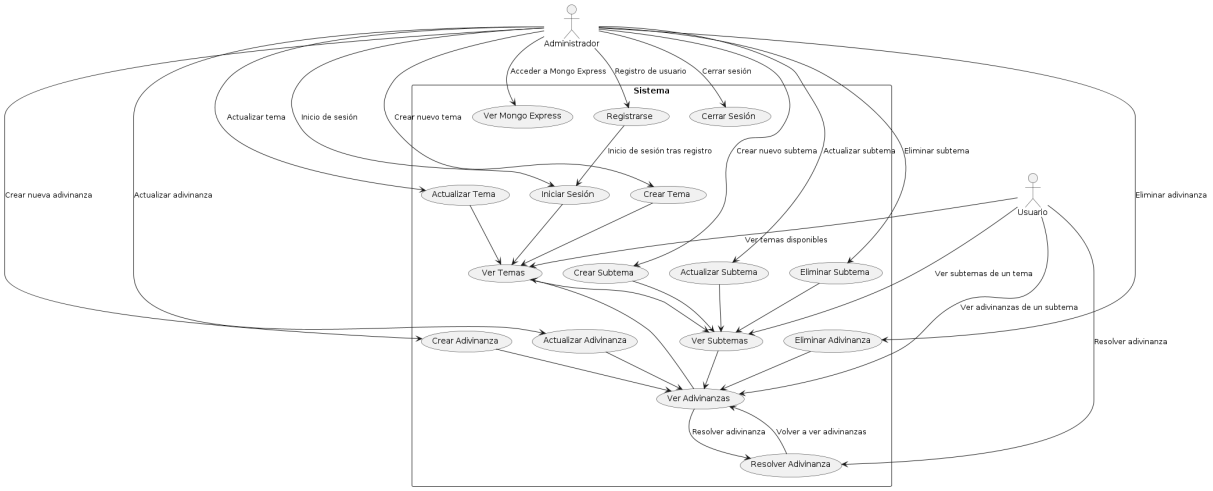

Figura 2: Caso de uso backend

```
@startuml Caso de uso Backend

actor Usuario as U
actor Administrador as A

rectangle Sistema {
  (Registrarse) as registrarse
  (Iniciar Sesión) as iniciarSesion
  (Cerrar Sesión) as cerrarSesion
  (Ver Temas) as verTemas
  (Crear Tema) as crearTema
  (Actualizar Tema) as actualizarTema
  (Ver Subtemas) as verSubtemas
  (Crear Subtema) as crearSubtema
  (Actualizar Subtema) as actualizarSubtema
  (Eliminar Subtema) as eliminarSubtema
  (Ver Adivinanzas) as verAdivinanzas
  (Crear Adivinanza) as crearAdivinanza
  (Actualizar Adivinanza) as actualizarAdivinanza
  (Eliminar Adivinanza) as eliminarAdivinanza
  (Resolver Adivinanza) as resolverAdivinanza
  (Ver Mongo Express) as verMongoExpress
}

U --> verTemas : "Ver temas disponibles"
U --> verSubtemas : "Ver subtemas de un tema"
```

```
U --> verAdivinanzas : "Ver adivinanzas de un subtema"
U --> resolverAdivinanza : "Resolver adivinanza"

A --> registrarse : "Registro de usuario"
A --> iniciarSesion : "Inicio de sesión"
A --> cerrarSesion : "Cerrar sesión"
A --> crearTema : "Crear nuevo tema"
A --> actualizarTema : "Actualizar tema"
A --> crearSubtema : "Crear nuevo subtema"
A --> actualizarSubtema : "Actualizar subtema"
A --> eliminarSubtema : "Eliminar subtema"
A --> crearAdivinanza : "Crear nueva adivinanza"
A --> actualizarAdivinanza : "Actualizar adivinanza"
A --> eliminarAdivinanza : "Eliminar adivinanza"
A --> verMongoExpress : "Acceder a Mongo Express"

registrarse --> iniciarSesion : "Inicio de sesión tras registro"
iniciarSesion --> verTemas
verTemas --> verSubtemas
verSubtemas --> verAdivinanzas
verAdivinanzas --> resolverAdivinanza : "Resolver adivinanza"
resolverAdivinanza --> verAdivinanzas : "Volver a ver adivinanzas"
verAdivinanzas --> verTemas

crearTema --> verTemas
actualizarTema --> verTemas
crearSubtema --> verSubtemas
actualizarSubtema --> verSubtemas
eliminarSubtema --> verSubtemas
crearAdivinanza --> verAdivinanzas
actualizarAdivinanza --> verAdivinanzas
eliminarAdivinanza --> verAdivinanzas

@enduml
```

1. **Usuarios**:
   - **Usuario** puede:
     - Ver temas disponibles
     - Ver subtemas de un tema
     - Ver adivinanzas de un subtema
     - Resolver adivinanzas
2. **Administradores**:
   - **Administrador** puede:
     - Registrar un usuario
     - Iniciar sesión

  - Cerrar sesión
  - Crear, actualizar y eliminar temas, subtemas y adivinanzas
  - Acceder a Mongo Express

3. **Sistema**:

   - Gestiona las actividades de registro, inicio y cierre de sesión.
   - Permite la visualización, creación, actualización y eliminación de temas, subtemas y adivinanzas.
   - Facilita a los usuarios la visualización de temas, subtemas y adivinanzas, así como la resolución de estas últimas.

9.2 Frontend

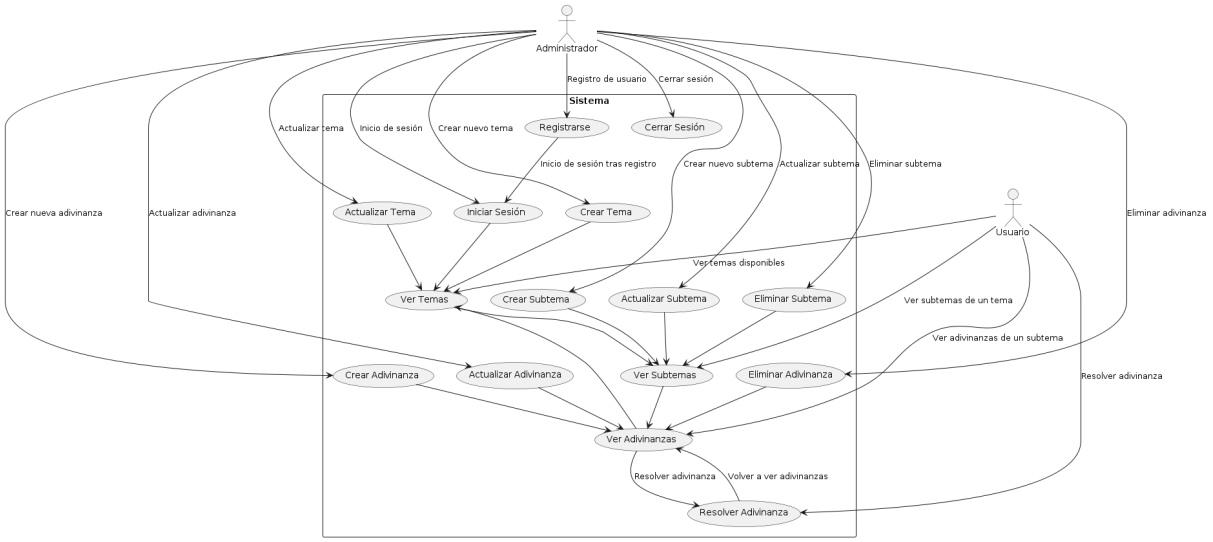

Figura 3: Caso de uso frontend

```
@startuml

actor Usuario as U
actor Administrador as A

rectangle Sistema {
  (Registrarse) as registrarse
  (Iniciar Sesión) as iniciarSesion
  (Cerrar Sesión) as cerrarSesion
  (Ver Temas) as verTemas
  (Crear Tema) as crearTema
  (Actualizar Tema) as actualizarTema
```

```
  (Ver Subtemas) as verSubtemas
  (Crear Subtema) as crearSubtema
  (Actualizar Subtema) as actualizarSubtema
  (Eliminar Subtema) as eliminarSubtema
  (Ver Adivinanzas) as verAdivinanzas
  (Crear Adivinanza) as crearAdivinanza
  (Actualizar Adivinanza) as actualizarAdivinanza
  (Eliminar Adivinanza) as eliminarAdivinanza
  (Resolver Adivinanza) as resolverAdivinanza
}

U --> verTemas : "Ver temas disponibles"
U --> verSubtemas : "Ver subtemas de un tema"
U --> verAdivinanzas : "Ver adivinanzas de un subtema"
U --> resolverAdivinanza : "Resolver adivinanza"

A --> registrarse : "Registro de usuario"
A --> iniciarSesion : "Inicio de sesión"
A --> cerrarSesion : "Cerrar sesión"
A --> crearTema : "Crear nuevo tema"
A --> actualizarTema : "Actualizar tema"
A --> crearSubtema : "Crear nuevo subtema"
A --> actualizarSubtema : "Actualizar subtema"
A --> eliminarSubtema : "Eliminar subtema"
A --> crearAdivinanza : "Crear nueva adivinanza"
A --> actualizarAdivinanza : "Actualizar adivinanza"
A --> eliminarAdivinanza : "Eliminar adivinanza"

registrarse --> iniciarSesion : "Inicio de sesión tras registro"
iniciarSesion --> verTemas
verTemas --> verSubtemas
verSubtemas --> verAdivinanzas
verAdivinanzas --> resolverAdivinanza : "Resolver adivinanza"
resolverAdivinanza --> verAdivinanzas : "Volver a ver adivinanzas"
verAdivinanzas --> verTemas

crearTema --> verTemas
actualizarTema --> verTemas
crearSubtema --> verSubtemas
actualizarSubtema --> verSubtemas
eliminarSubtema --> verSubtemas
crearAdivinanza --> verAdivinanzas
actualizarAdivinanza --> verAdivinanzas
eliminarAdivinanza --> verAdivinanzas

@enduml
```

9.2.1 Actores

- **Usuario**: Interactúa con las funcionalidades de ver y resolver.
- **Administrador**: Tiene acceso completo a las funcionalidades de gestión y administración.

9.2.1.1 Funcionalidades

- **Registrarse**: Permite a un nuevo administrador registrarse en el sistema.
- **Iniciar Sesión**: Permite a los administradores autenticarse en el sistema.
- **Cerrar Sesión**: Permite a los administradores cerrar su sesión.
- **Ver Temas**: Permite ver la lista de temas disponibles.
- **Crear Tema**: Permite a los administradores crear un nuevo tema.
- **Actualizar Tema**: Permite a los administradores actualizar un tema existente.
- **Ver Subtemas**: Permite ver la lista de subtemas dentro de un tema.
- **Crear Subtema**: Permite a los administradores crear un nuevo subtema.
- **Actualizar Subtema**: Permite a los administradores actualizar un subtema existente.
- **Eliminar Subtema**: Permite a los administradores eliminar un subtema existente.
- **Ver Adivinanzas**: Permite ver la lista de adivinanzas dentro de un subtema.
- **Crear Adivinanza**: Permite a los administradores crear una nueva adivinanza.
- **Actualizar Adivinanza**: Permite a los administradores actualizar una adivinanza existente.
- **Eliminar Adivinanza**: Permite a los administradores eliminar una adivinanza existente.
- **Resolver Adivinanza**: Permite a los usuarios resolver una adivinanza específica.