Documentación Proyecto IA y BIG DATA
Generador de ejercicios

- ## Propósito general

El propósito general del proyecto sería ayudar tanto a profesores como a estudiantes a la creación de actividades prácticas para los cursos de DAM, DAW y ASIR, con diferentes temas y niveles de dificultad, mejorando tanto la eficiencia de la enseñanza como los resultados de aprendizaje de los estudiantes.

- ## Metodología a seguir

La metodología que utilizaremos será la metodología Scrum, con sprints semanales, para organizar el trabajo de una manera ágil y sencilla.

1. Repartir el trabajo a realizar:
   Pensar cómo queremos hacer todo el proyecto y que estructura seguir, y después cada miembro se encargará de diferentes tareas para así agilizar el trabajo.

2. Crear Backend:
   Empezar con el servidor utilizando node.js y express, definiendo los endpoints y configurando los ficheros necesarios.

3. Base de datos:
   Crear la estructura de la base de datos utilizando mongoDB, para poder guardar los usuarios que se registren.

4. Crear Frontend:
   Desarrollo del frontend, diseñar la parte gráfica de la aplicación utilizando Flet.

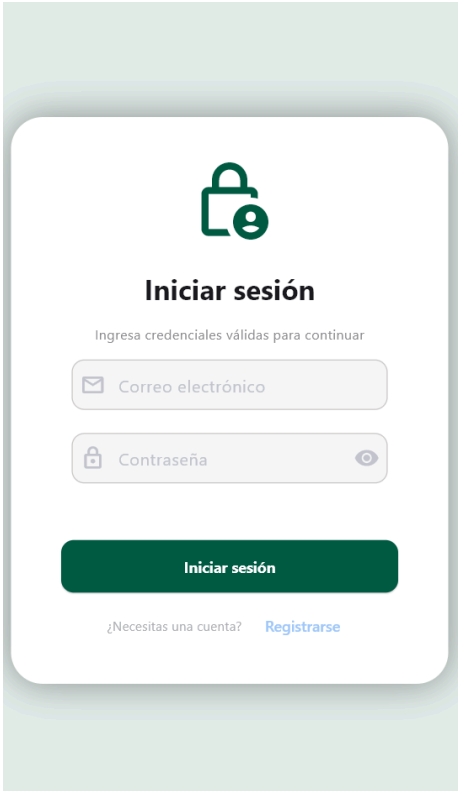
- **Tecnologías utilizadas**

- El frontend de la aplicación se desarrollará con Flet.
- El backend se implementará con Node.js.
- El componente de IA se basará en Amazon Bedrock, responsable de generar ejercicios educativos basados en los parámetros seleccionados. La conexión a este servicio se realizará mediante boto3 para gestionar y ejecutar solicitudes de IA.

- **Funcionalidades / páginas :**

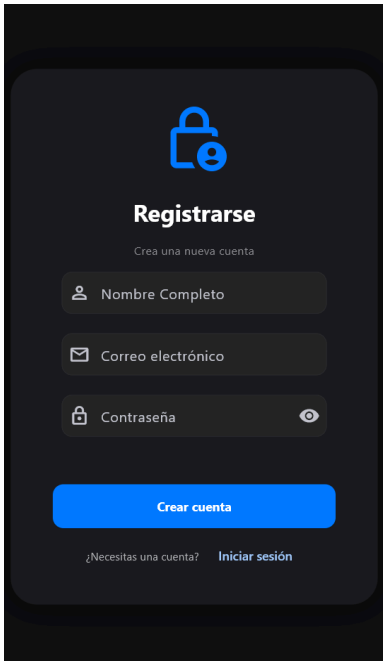
1. **página de Login:**

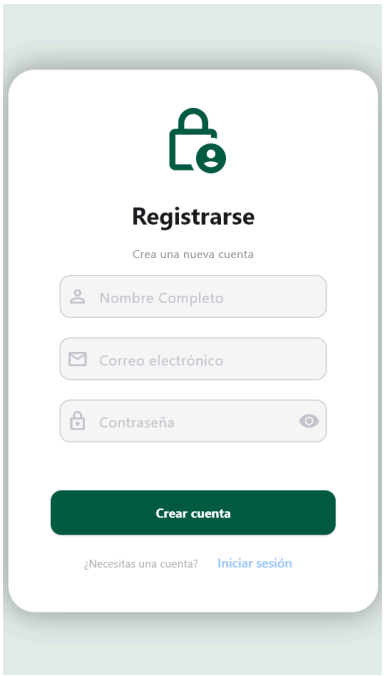
La página de inicio de sesión permite a los usuarios acceder a sus cuentas ingresando sus credenciales (correo electrónico de usuario y contraseña), incluye opciones para registrarse para nuevos usuarios.



2. página de Sign up:

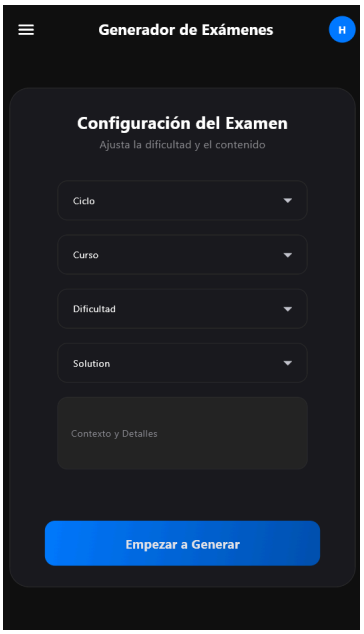
La página de registro permite a los nuevos usuarios crear una cuenta proporcionando la información necesaria, como correo electrónico, nombre de usuario y contraseña, A menudo incluye validación para garantizar la integridad de los datos y medidas de seguridad

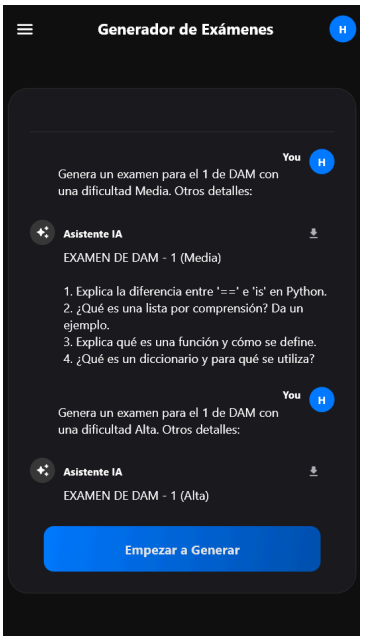


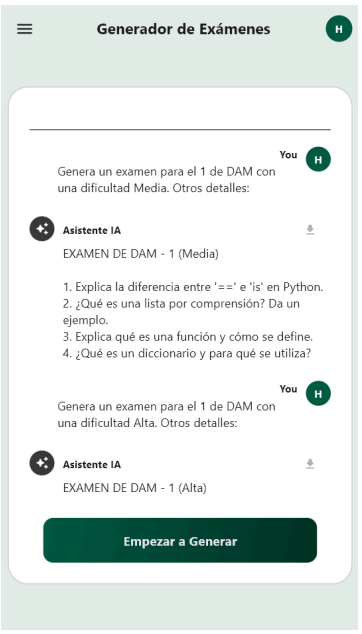


3. página de Inicio:

La página inicial permite crear exámenes para los cursos de grado superior de Informática, utilizando la IA generativa de Amazon Bedrock.

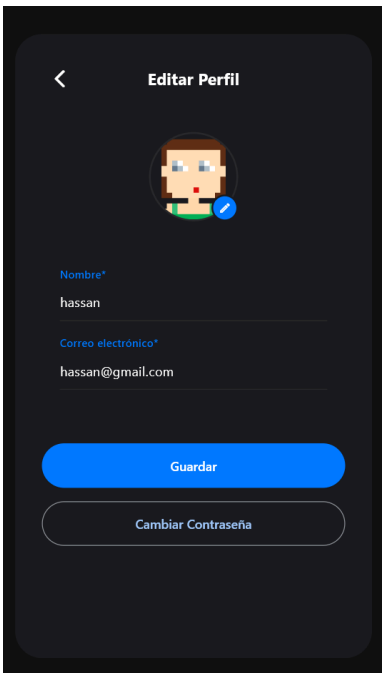


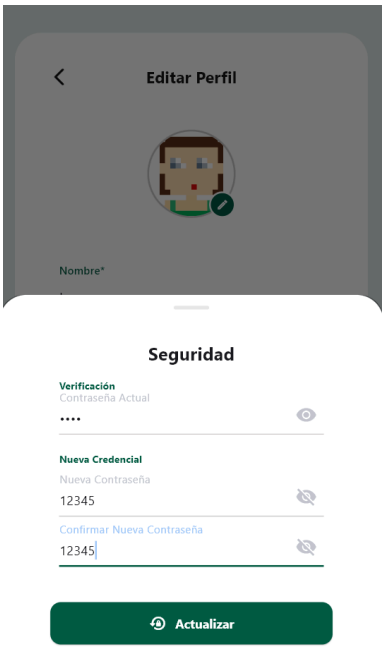


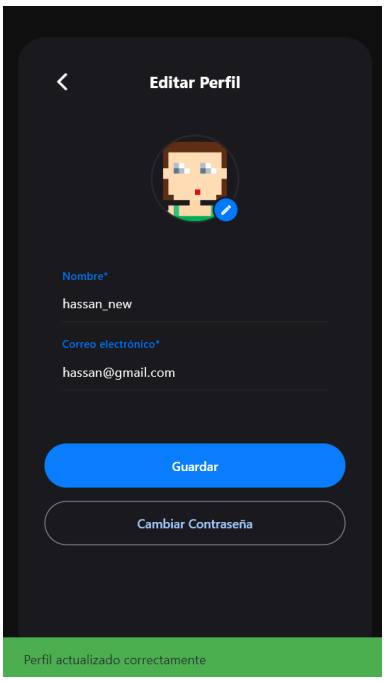


4. página de Perfil:

La página de perfil permite a los usuarios la opción de cambiar su información personal, como nombre de usuario,
dirección de correo electrónico y también pueden cambiar su contraseña. A menudo incluye validación para garantizar la integridad de los datos y medidas de seguridad

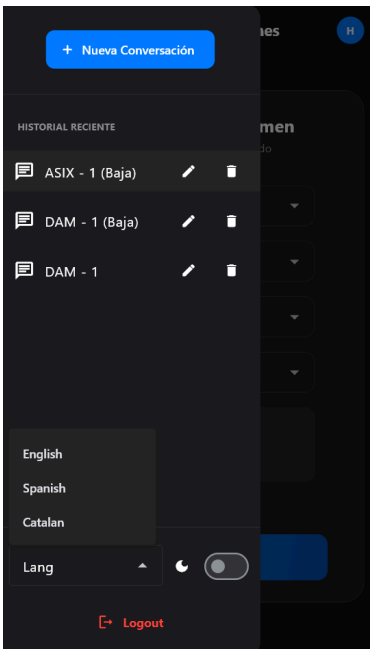


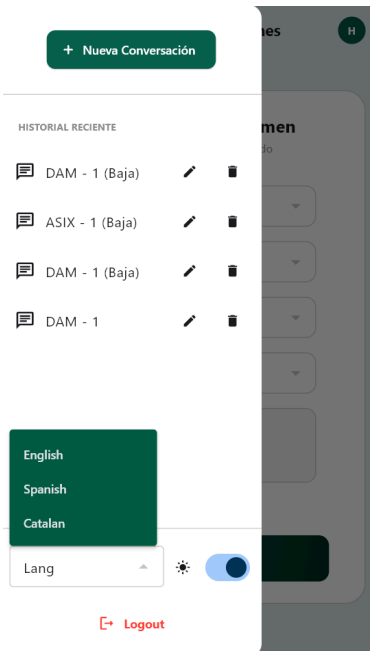


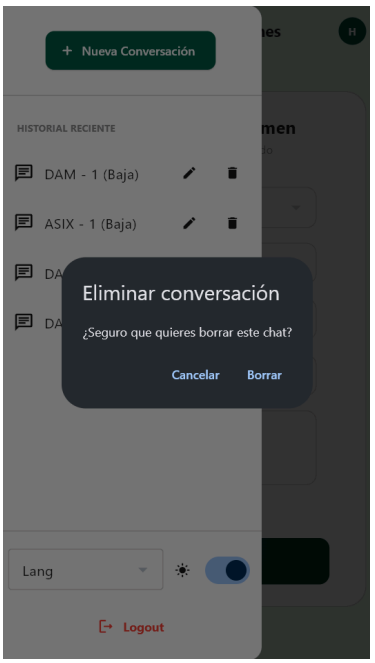


5. menú de la Barra lateral:

La barra lateral permite crear nuevas conversaciones para generar nuevos exámenes para otros ciclos.







Todas las conversaciones anteriores también se encuentran en la sección de historial. Hay dos opciones: editar el nombre de la conversación o eliminarla. La última sección permite cambiar el idioma a inglés, español o catalán, así como activar el modo oscuro y cerrar sesión.

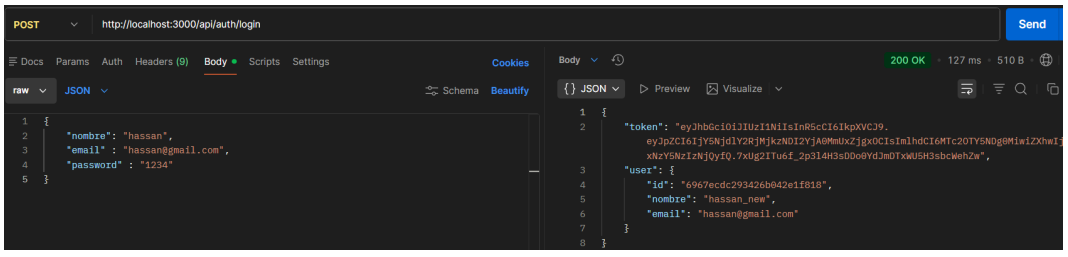
- **Ejemplo de Función en flet:**

el Flet, es fundamental estructurar bien las funciones para manejar correctamente las solicitudes y respuestas, especialmente cuando interactuamos con APIs. a continuación, se muestra un ejemplo de cómo implementar una función para modificar el nombre y correo electrónico

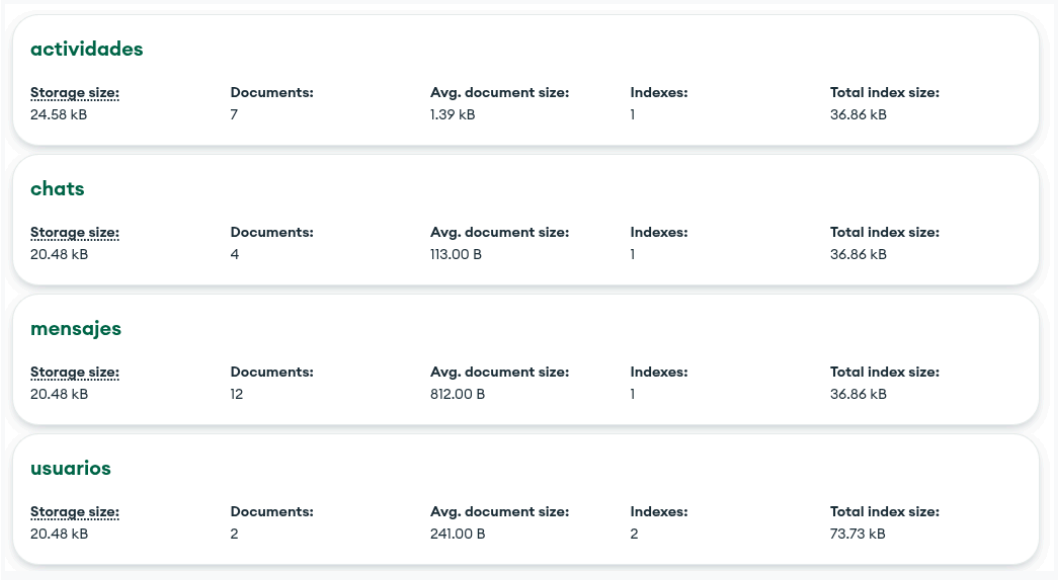
```python
async def save_profile(e):
        payload = {
            "id": user_id,
            "nombre": name_input.value,
            "email": email_input.value
        }

        try:
            response = requests.put(f"{BASE_URL}/update-profile", json=payload)

            if response.status_code == 200:
                data = response.json()
                page.session.store.set("user_name", data['user']['nombre'])
                page.session.store.set("user_email", data['user']['email'])

                snack_bar = ft.SnackBar(
                    ft.Text(t['profile']['profile_updated']),
                    bgcolor="green"
                )
            else:
                snack_bar = ft.SnackBar(
                    ft.Text(t['profile']['save_error']),
                    bgcolor="red"
                )

            page.overlay.append(snack_bar)
            snack_bar.open=True
            page.update()

        except Exception as err:
            print(f"Save Error: {err}")
```

- **Prueba de API:**

Para comprobar que todo funciona correctamente usaremos Postman que nos permite visualizar las respuestas.



- **Estructura de la base de datos:**



actividades

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	7	1.39 kB	1	36.86 kB

chats

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	4	113.00 B	1	36.86 kB

mensajes

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	12	812.00 B	1	36.86 kB

usuarios

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	2	241.00 B	2	73.73 kB

- **Conexión Backend-Base de datos:**

```
mongoose.connect(process.env.MONGO_URI)
  .then(() => {
    app.listen(3000, () => {
      console.log("Servidor escuchando en http://localhost/3000:");
    });
  });
```

- **Endpoints del backend:**
  1. **Ruta para registrar usuario**

     La utilizamos para poder crear un usuario dentro de la aplicación.

```
const router = express.Router();
router.post("/register", async (req, res) => {
  try {
    const { nombre, email, password } = req.body;

    if (!nombre || !email || !password) {
      return res.status(400).json({ error: "Faltan campos obligatorios" });
    }

    const existingUser = await Usuario.findOne({ email });
    if (existingUser) {
      return res.status(409).json({ error: "El email ya está registrado" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const user = await Usuario.create({
      nombre,
      email,
      password: hashedPassword
    });

    const token = jwt.sign(
      { id: user._id },
      process.env.JWT_SECRET,
      { expiresIn: "8h" }
    );

    res.status(201).json({
      token,
      user: {
        id: user._id,
        nombre: user.nombre,
        email: user.email
      }
    });
```

2. Ruta para iniciar sesión

La utilizamos para poder entrar a la aplicación con nuestro usuario.

```
router.post("/login", async (req, res) => {
  const { email, password } = req.body;

  //Comprobar usuario
  const user = await Usuario.findOne({ email });
  if (!user) {
    return res.status(401).json({ error: "Credenciales incorrectas" });
  }

  //Comprobar contraseña
  const valid = await bcrypt.compare(password, user.password);
  if (!valid) {
    return res.status(401).json({ error: "Credenciales incorrectas" });
  }

  //Generar JWT
  const token = jwt.sign(
    { id: user._id },
    process.env.JWT_SECRET,
    { expiresIn: "8h" }
  );

  res.json({
    token,
    user: {
      id: user._id,
      nombre: user.nombre,
      email: user.email
    }
  });
});
```

3. Ruta para modificar usuario y contraseña

Para que el usuario pueda modificar el usuario a su gusto

```
router.put("/update", authenticateToken, async (req, res) => {
  try {
    const { nombre, email, avatar } = req.body;

    // Validar campos obligatorios
    if (!nombre || !email) {
      return res.status(400).json({ error: "Faltan campos obligatorios" });
    }

    // Buscar usuario autenticado
    const user = await Usuario.findById(req.user.id);
    if (!user) {
      return res.status(404).json({ error: "Usuario no encontrado" });
    }

    // Verificar si el email ya está en uso por otro usuario
    if (email !== user.email) {
      const emailExists = await Usuario.findOne({
        email,
        _id: { $ne: req.user.id }
      });
      if (emailExists) {
        return res.status(400).json({ error: "El email ya está en uso" });
      }
    }

    // Actualizar campos
    const updateData = {
      nombre,
      email
    };

    if (avatar) {
      updateData.avatar = avatar;
    }

    const updatedUser = await Usuario.findByIdAndUpdate(
      req.user.id,
      { $set: updateData },
      { new: true, runValidators: true }
    ).select("-password");

    res.json({
      message: "Perfil actualizado correctamente",
      user: updatedUser
    });
```

```
router.put("/cambiar-password", authenticateToken, async (req, res) => {
  try {
    const { currentPassword, newPassword } = req.body;

    // Validar campos obligatorios
    if (!currentPassword || !newPassword) {
      return res.status(400).json({ error: "Faltan campos obligatorios" });
    }

    // Buscar usuario
    const user = await Usuario.findById(req.user.id);
    if (!user) {
      return res.status(404).json({ error: "Usuario no encontrado" });
    }

    // Verificar contraseña actual
    const isPasswordValid = await bcrypt.compare(currentPassword, user.password);
    if (!isPasswordValid) {
      return res.status(400).json({ error: "Contraseña actual incorrecta" });
    }

    // Encriptar nueva contraseña
    const hashedPassword = await bcrypt.hash(newPassword, 10);

    // Actualizar contraseña
    await Usuario.findByIdAndUpdate(req.user.id, {
      password: hashedPassword
    });

    res.json({
      message: "Contraseña actualizada correctamente"
    });

  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Error al cambiar la contraseña" });
  }
});
```

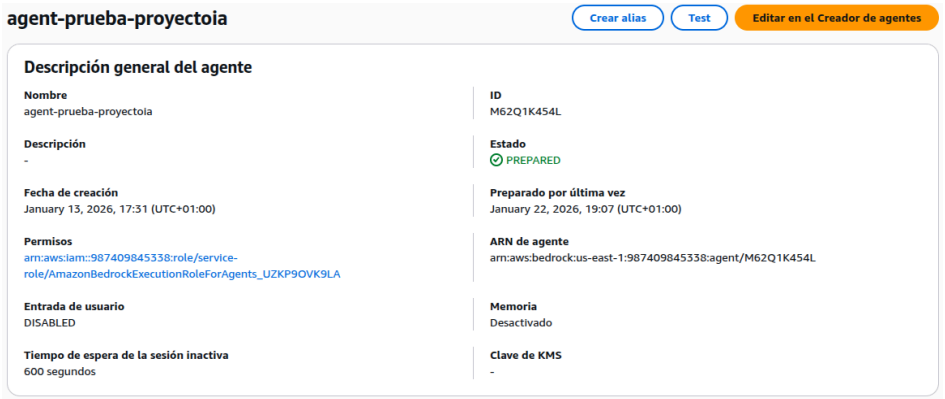
4. Ruta para generar los exámenes

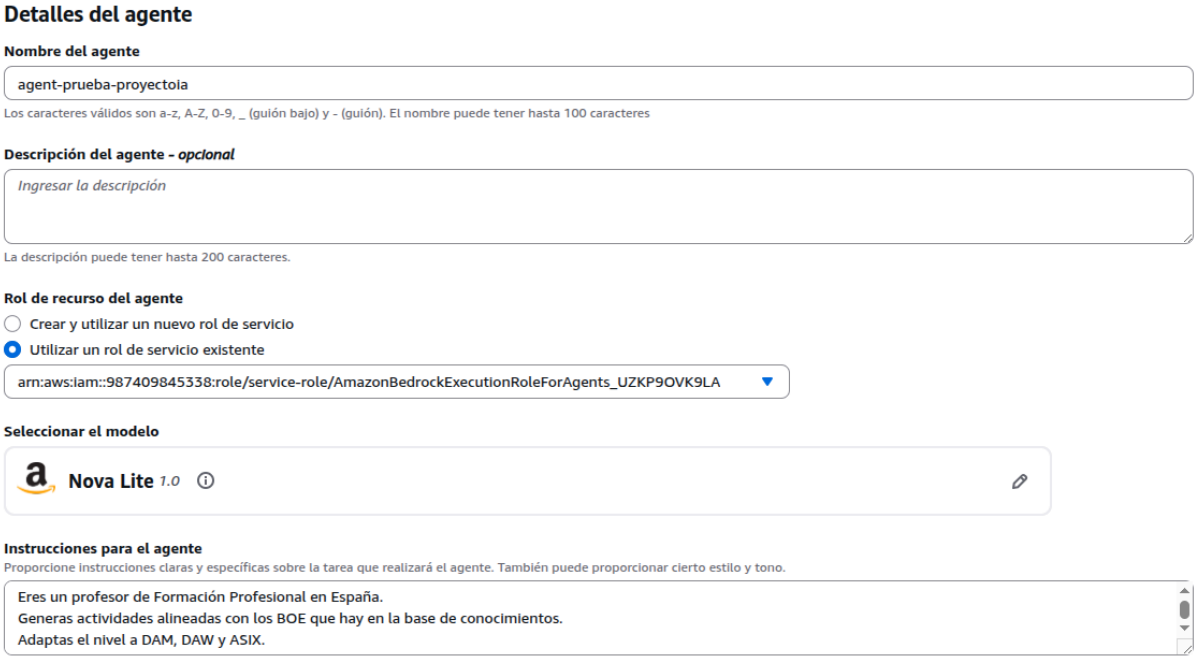
```
router.post("/generate-ia", authenticateToken, async (req, res) => {
  try {
    const { ciclo, curso, nivel, solucion, userPrompt, chatId } = req.body;

    if (!ciclo || !curso || !nivel || !solucion || !userPrompt) {
      return res.status(400).json({ error: "Faltan campos obligatorios" });
    }

    //Crear o reutilizar chat
    let chat;

    if (chatId) {
      chat = await Chat.findOne({ _id: chatId, userId: req.user.id });
      if (!chat) {
        return res.status(404).json({ error: "Chat no encontrado" });
      }
    } else {
      chat = await Chat.create({
        userId: req.user.id,
        title: `${ciclo} - ${curso} (${nivel})`
      });
    }

    //Guardar mensaje del usuario
    await Mensajes.create({
      chatId: chat._id,
      userId: req.user.id,
      role: "user",
      message: userPrompt
    });

    const prompt = `
        Rol del usuario: ${req.user.rol}

        Ciclo: ${ciclo}
        Curso: ${curso}
        Nivel: ${nivel}
```

- ### Agente de Amazon Bedrock

  Hemos creado un agente utilizando la tecnología de Bedrock, configurandolo con unas instrucciones específicas para que conteste como se desea.

  También el agente utiliza una base de conocimientos donde están los BOE de los ciclos superiores de DAM, DAW y ASIX.





- **Miembros del equipo**

- Jose Adrián Aglio Gascón
- Alhassan Bouhou