BossFlow

Requisitos técnicos

Backend:

- **Endpoints utilizados:**
  - GET:
    - GET /
    - GET /health
    - GET /perfil
    - GET /profile
    - GET /profile/stats
    - GET /diagrams
    - GET /diagrams/:id
    - GET /templates
  - POST:
    - POST /auth/register
    - POST /auth/login
    - POST /auth/logout
    - POST /diagrams
    - POST /images/upload
    - POST /images/validate-url
    - POST /eco
  - PUT:
    - PUT /profile
    - PUT /diagrams/:id
  - DELETE:
    - DELETE /diagrams/:id
    - DELETE /images
- **Errores:**
  - Validación de campos
  - Validación del formato del email
  - Validación longitud usuario
  - Validación longitud contraseña
  - Validación duplicidad de email
  - Validación duplicidad de usuario
  - Validación credenciales
- **Operaciones protegidas:**
  - GET /perfil
  - GET /profile
  - GET /diagrams

Base de datos:

- **Mongoose:**

backend > models > Diagram.js > ...

```
const mongoose = require('mongoose');

// Esquema de metadata de imagen
const ImageMetadataSchema = new mongoose.Schema({
    filename: {
        type: String,
        required: true,
        trim: true
```

backend > models > User.js > ...

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const validator = require('validator');

const UserSchema = new mongoose.Schema({
  username: {
        type: String,
        required: true,
```

backend > config > database.js > ...

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI).then(
      console.log('✅ MongoDB conectado correctamente'
    });
  } catch (error) {
```

Frontend:

- **SPA:**

```jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import { ToastProvider } from './context/ToastContext';
import PrivateRoute from './routes/PrivateRoute';
import Home from './pages/Home';
import Dashboard from './pages/Dashboard';
import Templates from './pages/Templates';
import Editor from './pages/Editor';
import Status from './pages/Status';
import Community from './pages/Community';
import Profile from './pages/Profile';
import NotFound from './pages/NotFound';
import Login from './components/Login/Login';
import Register from './components/Register/Register';
import Layout from './layouts/Layout';

function App() {
  return (
    <AuthProvider>
      <ToastProvider>
        <Router>
          <Routes>
            <Route element={<Layout />}>
              {/* Rutas públicas */}
              <Route path="/" element={<Home />} />
              <Route path="/status" element={<Status />} />
              <Route path="/community" element={<Community />} />
              <Route path="/login" element={<Login />} />
              <Route path="/register" element={<Register />} />

              {/* Rutas privadas */}
              <Route path="/editor/:diagramId" element={
                <PrivateRoute>
                  <Editor />
                </PrivateRoute>} />

              <Route path="/profile" element={
                <PrivateRoute>
                  <Profile />
                </PrivateRoute>} />

              <Route path="/dashboard" element={
                <PrivateRoute>
                  <Dashboard />
                </PrivateRoute>} />

              <Route path="/dashboard/plantillas" element={
                <PrivateRoute>
                  <Templates />
                </PrivateRoute>} />

              {/* Ruta 404 - debe estar al final */}
              <Route path="*" element={<NotFound />} />
            </Route>
          </Routes>
        </Router>
      </ToastProvider>
    </AuthProvider>
  );
}

export default App;
```

- **PrivateRoute:**

frontend > src > routes > PrivateRoute.jsx > PrivateRoute

```jsx
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
import './PrivateRoute.css';

/**
 * Componente para proteger rutas que requieren autenticación
 *
 * Uso:
 * <Route
 *   path="/ruta-protegida"
 *   element={
 *     <PrivateRoute>
 *       <ComponenteProtegido />
 *     </PrivateRoute>
 *   }
 * />
 */
function PrivateRoute({ children }) {
  const { isAuthenticated, loading } = useAuth();

  // Mientras verifica autenticación, mostrar loading
  if (loading) {
    return (
      <div className="private-route__loading">
        <div className="private-route__spinner"></div>
        <p className="private-route__texto">Verificando autenticación...</p>
      </div>
    );
  }

  // Si no está autenticado, redirigir a login
  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  // Si está autenticado, renderizar el componente protegido
  return children;
}

export default PrivateRoute;
```

- **Validaciones:**

frontend > src > components > Login > Login.jsx > ...

```
7   function Login() {
48
49    // Validar formulario
50    const validarFormulario = () => {
51      const nuevosErrores = {};
52
53      // Validar correo
54      if (!datosFormulario.correo) {
55        nuevosErrores.correo = 'Es obligatorio introducir un correo electrónico.';
56      } else if (
57        !/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/.test(
58          datosFormulario.correo
59        )
60      ) {
61        nuevosErrores.correo = 'El correo electrónico no es válido.';
62      }
63
64      // Validar contraseña
65      if (!datosFormulario.contrasena) {
66        nuevosErrores.contrasena = 'Es obligatorio introducir una contraseña.';
67      } else if (datosFormulario.contrasena.length < 8) {
68        nuevosErrores.contrasena =
69          'La contraseña debe tener al menos 8 caracteres.';
70      }
71
72      return nuevosErrores;
73    };
74
75    // Control de submits
76    const manejoSubmit = async (e) => {
77      e.preventDefault();
78
79      const nuevosErrores = validarFormulario();
80      if (Object.keys(nuevosErrores).length > 0) {
81        setErrores(nuevosErrores);
82        return;
83      }
84
85      try {
86        setErrores({});
87        setCargando(true);
88
89        // Llamar a la función login del contexto
90        const resultado = await login(
91          datosFormulario.correo,
```

- **Eventos:**

```
const alternarMenu = () => {
    setMenuAbierto(!menuAbierto);
};

<button
    className="navbar__usuario"
    onClick={alternarMenu}
>
    {/* contenido */}
</button>
```

En Dashboard (Cerrar sesión):

```
const handleLogout = () => {
    logout();
    navigate('/');
};

<button
    className="sidebar__logout"
    onClick={handleLogout}
>
    Cerrar sesión
</button>
```

Interacción entre frontend y backend:

- **Flujo frontend/backend con Axios:**

frontend > src > services > api.js > ...

```
import axios from 'axios';

// En desarrollo: si defines VITE_API_URL (por ejemplo http://localhost:5000)
// se usará como base absoluta. En producción queremos rutas relativas
// para que el proxy (Nginx) maneje /api. Por eso la convención:
// - VITE_API_URL= (no definido)  -> base '/api' (rutas relativas al host)
// - VITE_API_URL=http://host:port -> base absoluta hacia el backend
const baseURL = import.meta.env.VITE_API_URL || '/api';

const api = axios.create({
  baseURL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

api.interceptors.request.use(
  (config) => {
    // Agregar token de autenticación si existe
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

api.interceptors.response.use(
  (response) => response,
  (error) => {
    // Detectar si el token ha expirado (401 Unauthorized)
    if (error.response && error.response.status === 401) {
      const errorMessage = error.response.data?.error || '';

      // Si el error indica que el token es inválido o expirado
      if (errorMessage.includes('Token inválido') || errorMessage.includes('expirado')) {
        // Limpiar localStorage
        localStorage.removeItem('token');
        localStorage.removeItem('user');

        // Emitir evento personalizado para que AuthContext lo detecte
        window.dispatchEvent(new Event('token-expired'));

        console.warn('Token expirado. Sesión cerrada automáticamente.');
      }
```

- **Zustand (No utilizado, simplemente hecho con Context API):**

frontend > src > context > AuthContext.jsx > ...

```jsx
import { createContext, useState, useContext, useEffect } from 'react';
import api from '../services/api';
import authService from '../services/authService';

// Crear el contexto
const AuthContext = createContext(null);

// Hook personalizado para usar el contexto
export const useAuth = () => {
    const context = useContext(AuthContext);
    if (!context) {
        throw new Error('useAuth debe ser usado dentro de un AuthProvider');
    }
    return context;
};

// Provider del contexto
export const AuthProvider = ({ children }) => {
    const [user, setUser] = useState(null);
    const [isAuthenticated, setIsAuthenticated] = useState(false);
    const [loading, setLoading] = useState(true);

    // Verificar si hay un usuario autenticado al cargar la aplicación
    useEffect(() => {
        checkAuth();

        // Escuchar evento de token expirado
        const handleTokenExpired = () => {
            console.log('Token expirado detectado, cerrando sesión...');
            logout();
            // Redirigir al login
            window.location.href = '/login';
        };

        window.addEventListener('token-expired', handleTokenExpired);

        // Cleanup
        return () => {
            window.removeEventListener('token-expired', handleTokenExpired);
        };
    }, []);

    // Verificar autenticación desde localStorage
    const checkAuth = async () => {
        try {
            const storedUser = localStorage.getItem('user');
            const token = localStorage.getItem('token');

            if (storedUser && token) {
```

Seguridad:

backend > models > User.js > ...

```
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcrypt');
3  const validator = require('validator');
```

backend > controllers > authController.js > register > register

```
1  const User = require('../models/User');
2  const jwt = require('jsonwebtoken');
3  const validator = require('validator');
4  const JWT_SECRET = process.env.JWT_SECRET
5
6  exports.register = async (req, res, next) => {
7      try {
8          const { username, email, password, rememberMe } = req.body;
9
10         // Valida que todos los campos requeridos estén presentes
11         if (!username || !email || !password) {
12             return res.status(400).json({
13                 error: 'Todos los campos son requeridos (username, email, password)'
14             });
15         }
16
17         // Valida formato del email
18         if (!validator.isEmail(email)) {
19             return res.status(400).json({
20                 error: 'El formato del email no es válido'
21             });
22         }
23
24         // Valida longitud mínima del username
25         if (username.trim().length < 3) {
26             return res.status(400).json({
27                 error: 'El username debe tener al menos 3 caracteres'
28             });
29         }
30
31         // Valida longitud mínima del password
32         if (password.length < 8) {
33             return res.status(400).json({
34                 error: 'La contraseña debe tener al menos 8 caracteres'
35             });
36         }
37
38         // Verifica que el email no esté ya registrado
39         const existingEmail = await User.findOne({ email });
40         if (existingEmail) {
41             return res.status(400).json({
42                 error: 'El email ya está registrado'
43             });
44         }
45
46         // Verifica que el username no esté ya en uso
47         const existingUsername = await User.findOne({ username });
48         if (existingUsername) {
49             return res.status(400).json({
```

backend > middleware > auth.js > ...

```js
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET;

function auth(req, res, next) {
    // Obtiene el header Authorization
    const header = req.headers.authorization;

    // Comprueba que exista y tenga formato Bearer
    if (!header || !header.startsWith("Bearer ")) {
        return res.status(401).json({ error: "Token requerido" });
    }

    // Extrae el token del header
    const token = header.split(" ")[1];

    try {
        // Verifica y decodifica el token
        const decoded = jwt.verify(token, JWT_SECRET);

        // Guarda los datos del usuario en la request
        req.user = decoded;

        // Continúa con la ruta protegida
        next();

    } catch (err) {
        // Token inválido o expirado
        return res.status(401).json({ error: "Token inválido o expirado" });
    }
}

module.exports = auth;
```

backend > routes > index.js > ...

```js
const express = require("express");
const router = express.Router();
const auth = require("../middleware/auth");
const authController = require("../controllers/authController");
const diagramController = require("../controllers/diagramController");
const profileController = require("../controllers/profileController");
const imageController = require("../controllers/imageController");

// Peticiones GET
router.get("/", (req, res) => {
  res.json({ mensaje: "✅API funcionando correctamente" });
});

router.get("/health", (req, res) => {
  res.json({
    status: "ok",
    timestamp: Date.now(),
  });
});

router.get("/perfil", auth, (req, res) => {
  res.json({user: req.user});
});

router.get("/profile", auth, (req, res, next) => {
  profileController.getProfile(req, res, next);
});

router.get("/profile/stats", auth, (req, res, next) => {
  profileController.getStats(req, res, next);
});

router.get("/diagrams", auth, (req, res, next) => {
  diagramController.getDiagrams(req, res, next);
});

router.get("/diagrams/:id", auth, (req, res, next) => {
  diagramController.getDiagramById(req, res, next);
});

router.get("/templates", auth, (req, res, next) => {
  diagramController.getTemplates(req, res, next);
});

```

Requisitos metodológicos

Organización y gestión:

Se ha utilizado SCRUM para el planteamiento del proyecto, cada uno de los tres integrantes ha tenido un rol diferente cada semana entre SCRUM MASTER, PRODUCT OWNER y DEVELOPER (aunque todos han sido developers). Las tareas están en el proyecto del repositorio, estando asignadas a cada integrante personalmente y pudiendo ver el estado de progreso en el que se encuentra. Todo el código se encuentra documentado con JSDoc.