Documentación Técnica: BossFlow

1. Documentación de la API

La API REST del backend está construida sobre Express.js y sigue principios RESTful con autenticación JWT. Todas las rutas están prefijadas con `/api`.

1.1. Endpoints de Autenticación

POST `/api/auth/register`: Registra un nuevo usuario en la aplicación.

Parámetros requeridos:

- Headers:
  Content-Type: application/json
- Body (JSON):
  {
  "username": "string (mínimo 3 caracteres)",
  "email": "string (formato email válido)",
  "password": "string (mínimo 8 caracteres)",
  "rememberMe": "boolean (opcional, default: false)"
  }

Ejemplos de respuestas:

- 201 Created
  {
  "message": "Usuario registrado exitosamente",
  "token": "jwt_token_string",
  "user": {
  "id": "ObjectId",
  "username": "username",
  "email": "email@example.com",
  "avatar": null
  }
  }
- 400 Bad Request
  {
  "error": "Todos los campos son requeridos (username, email, password)"
  }
- 409 Conflict
  {
  "error": "El email ya está registrado"
  }

POST `/api/auth/login`: Autentica un usuario existente y genera un token JWT.

Parámetros requeridos:

- Headers:
  Content-Type: application/json
- Body (JSON):
  {
  "email": "string (formato email válido)",
  "password": "string",
  "rememberMe": "boolean (opcional, default: false)"
  }

Ejemplos de respuestas:

- 200 OK
  {
  "message": "Login exitoso",
  "token": "jwt_token_string",
  "user": {
  "id": "ObjectId",
  "username": "username",
  "email": "email@example.com",
  "avatar": null
  }
  }
- 400 Bad Request
  {
  "error": "Todos los campos son requeridos (email, password)"
  }
- 401 Unauthorized
  {
  "error": "Credenciales inválidas"
  }

POST `/api/auth/logout`: Cierra la sesión del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
  Content-Type: application/json

Ejemplos de respuestas:

- 200 OK
  {
  "message": "Sesión cerrada correctamente"
  }
- 401 Unauthorized
  {

"error": "Token requerido"
}

1.2. Endpoints de Diagramas

POST `/api/diagrams`: Crea un nuevo diagrama para el usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
  Content-Type: application/json
- Body (JSON):
  {
  "title": "string (mínimo 3 caracteres)",
  "description": "string (opcional, máximo 500 caracteres)",
  "nodes": [
  {
  "id": "string (único)",
  "type": "string (actionNode, decisionNode, bossNode, etc.)",
  "position": { "x": "number", "y": "number" },
  "data": { "label": "string", "description": "string (opcional)", "other_properties": "mixed" },
  "image": { "filename": "string", "url": "string", "mimeType": "string", "size": "number", "createdAt": "Date" }
  }
  ],
  "edges": [{ "id": "string", "source": "string", "target": "string" }],
  "isTemplate": "boolean (opcional, default: false)",
  "images": []
  }

Ejemplos de respuestas:

- 201 Created
  {
  "message": "Diagrama creado exitosamente",
  "diagram": { "id": "ObjectId", "title": "Título del diagrama", "nodes": [], "edges": [], "images": [] }
  }
- 400 Bad Request
  {
  "error": "El título es requerido y debe tener al menos 3 caracteres"
  }
- 409 Conflict
  {
  "error": "Ya existe un diagrama con ese título"
  }

GET `/api/diagrams`: Obtiene todos los diagramas del usuario autenticado (excluye plantillas).

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
- Query: ninguno

Ejemplos de respuestas:

- 200 OK
  {
  "diagrams": [
  { "id": "ObjectId", "title": "Título del diagrama", "nodes": [], "edges": [], "isTemplate": false }
  ]
  }
- 401 Unauthorized
  {
  "error": "Token requerido"
  }

GET `/api/diagrams/:id`: Obtiene un diagrama específico por su ID.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
- URL parameters:
  - `id`: ObjectId del diagrama

Ejemplos de respuestas:

- 200 OK
  {
  "diagram": { "id": "ObjectId", "title": "Título del diagrama", "nodes": [], "edges": [] }
  }
- 404 Not Found
  {
  "error": "Diagrama no encontrado o no autorizado"
  }

PUT `/api/diagrams/:id`: Actualiza un diagrama existente del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
  Content-Type: application/json
- URL parameters:
  - `id`: ObjectId del diagrama

- Body (JSON): campos opcionales (title, description, nodes, edges, images)

Ejemplos de respuestas:

- 200 OK
  {
  "message": "Diagrama actualizado exitosamente",
  "diagram": { "id": "ObjectId", "title": "Título actualizado" }
  }
- 400 Bad Request
  {
  "error": "El título debe tener al menos 3 caracteres"
  }
- 404 Not Found
  {
  "error": "Diagrama no encontrado o no autorizado"
  }

DELETE `/api/diagrams/:id`: Elimina un diagrama específico del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
- URL parameters:
  - `id`: ObjectId del diagrama

Ejemplos de respuestas:

- 200 OK
  {
  "message": "Diagrama eliminado exitosamente"
  }
- 404 Not Found
  {
  "error": "Diagrama no encontrado o no autorizado"
  }

GET `/api/templates`: Obtiene todas las plantillas del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>

Ejemplos de respuestas:

- 200 OK
  {
  "templates": [{ "id": "ObjectId", "title": "Título de la plantilla", "isTemplate": true }]
  }
- 401 Unauthorized
  {

"error": "Token requerido"
}

1.3. Endpoints de Perfil

GET `/api/perfil`: Obtiene los datos básicos del usuario autenticado (versión simplificada).

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>

Ejemplos de respuestas:

- 200 OK
  {
  "user": { "userId": "ObjectId", "iat": 1234567890, "exp": 1234567890 }
  }
- 401 Unauthorized
  {
  "error": "Token requerido"
  }

GET `/api/profile`: Obtiene el perfil completo del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>

Ejemplos de respuestas:

- 200 OK
  {
  "user": { "_id": "ObjectId", "username": "username", "email": "email@example.com",
  "avatar": "url_string" }
  }
- 404 Not Found
  {
  "error": "Usuario no encontrado"
  }

PUT `/api/profile`: Actualiza el perfil del usuario autenticado.

Parámetros requeridos:

- Headers:
  Authorization: Bearer <jwt_token>
  Content-Type: application/json
- Body (JSON): campos opcionales: `username`, `bio`, `favoriteGames`, `avatar`

Ejemplos de respuestas:

- 200 OK
  {

"message": "Perfil actualizado correctamente",
"user": { "_id": "ObjectId", "username": "nuevo_username", "email": "email@example.com" }
}

- 400 Bad Request
{
"error": "El nombre de usuario debe tener al menos 3 caracteres"
}

GET `/api/profile/stats`: Obtiene las estadísticas y logros del usuario autenticado.

Parámetros requeridos:

- Headers:
Authorization: Bearer <jwt_token>

Ejemplos de respuestas:

- 200 OK
{
"stats": { "diagramsCreated": 10, "nodesCreated": 150, "collaborations": 5 },
"achievements": [{ "name": "Primer Diagrama", "description": "Creaste tu primer diagrama" }]
}
- 401 Unauthorized
{
"error": "Token requerido"
}

1.4. Endpoints de Imágenes

POST `/api/images/upload`: Sube una imagen en formato base64 al servidor.

Parámetros requeridos:

- Headers:
Authorization: Bearer <jwt_token>
Content-Type: application/json
- Body (JSON):
{
"image": "string (base64 data o ...)",
"filename": "string (opcional)",
"mimeType": "string (image/jpeg, image/png, image/gif, image/webp)"
}

Ejemplos de respuestas:

- 201 Created
{
"message": "Imagen subida exitosamente",
"image": { "filename": "nombre_archivo.jpg", "url":

"/uploads/images/hash_unico.jpg", "mimeType": "image/jpeg", "size": 123456 }
}

- 400 Bad Request
{
"error": "No se proporcionó ninguna imagen"
}
- 401 Unauthorized
{
"error": "Token requerido"
}

POST `/api/images/validate-url`: Valida una URL de imagen externa.

Parámetros requeridos:

- Headers:
Authorization: Bearer <jwt_token>
Content-Type: application/json
- Body (JSON):
{ "url": "string (URL completa de la imagen)" }

Ejemplos de respuestas:

- 200 OK
{
"message": "URL validada exitosamente",
"image": { "filename": "nombre_extraido.jpg", "url":
"https://example.com/image.jpg", "mimeType": "image/jpeg", "size": 0 }
}
- 400 Bad Request
{ "error": "URL no válida" }
- 401 Unauthorized
{ "error": "Token requerido" }

DELETE `/api/images`: Elimina una imagen del servidor.

Parámetros requeridos:

- Headers:
Authorization: Bearer <jwt_token>
Content-Type: application/json
- Body (JSON):
{ "url": "string (URL de la imagen a eliminar)" }

Ejemplos de respuestas:

- 200 OK (imagen local eliminada)
{ "message": "Imagen eliminada exitosamente" }
- 200 OK (URL externa)
{ "message": "Imagen referenciada eliminada (URL externa)" }
- 400 Bad Request
{ "error": "No se proporcionó ninguna URL" }

1.5. Endpoints de Sistema

GET `/api/`: Verifica que la API está funcionando correctamente.

Parámetros requeridos: ninguno

Ejemplos de respuestas:

- 200 OK
  { "mensaje": "API funcionando correctamente" }

GET `/api/health`: Devuelve el estado actual del servidor y timestamp.

Parámetros requeridos: ninguno

Ejemplos de respuestas:

- 200 OK
  { "status": "ok", "timestamp": 1702303200000 }

POST `/api/eco`: Endpoint de prueba que devuelve el cuerpo de la solicitud enviada (solo en desarrollo).

Parámetros requeridos:

- Headers:
  Content-Type: application/json
- Body: cualquier JSON válido

Ejemplos de respuestas:

- 200 OK
  { "mismo_contenido_enviado": "..." }

Nota: Este endpoint solo funciona en modo desarrollo (`NODE_ENV !== production`).

2. Complejidad y Profundidad Técnica

2.1. Patrones de Diseño Implementados

Arquitectura MVC (Model-View-Controller)

El proyecto sigue estrictamente el patrón MVC separando responsabilidades:

- **Models** (`backend/models/`): Esquemas de Mongoose con validaciones, hooks y métodos personalizados
  - `User.js`: Modelo con hash automático de contraseñas usando bcrypt en pre-save hook
  - `Diagram.js`: Modelo complejo con subdocumentos anidados (nodes, edges, images)

- **Controllers** (`backend/controllers/`): Lógica de negocio desacoplada de las rutas
    - `authController.js`: Maneja registro, login y logout con validación exhaustiva
    - `diagramController.js`: CRUD completo con validaciones estructurales
    - `profileController.js`: Gestión de perfiles y estadísticas
    - `imageController.js`: Procesamiento de imágenes base64 y validación de URLs
- **Views/Routes** (`backend/routes/`): Definición de endpoints REST con middleware chain
    - Rutas claramente documentadas con JSDoc
    - Middleware de autenticación aplicado selectivamente

Middleware Chain Pattern

Uso sistemático de middleware de Express:

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

- **Autenticación** (`middleware/auth.js`): Verificación JWT con manejo de errores específico
- **Validación**: Validadores personalizados antes de la lógica de negocio
- **Error handling global**: Middleware de error centralizado en `server.js`

2.2. Buenas Prácticas Aplicadas

Seguridad

1. **Autenticación JWT robusta:**
    - Tokens con expiración configurable (7 días o 30 días con "Remember Me")
    - Secret key almacenada en variables de entorno
    - Verificación en cada request protegido
    - Header Authorization con formato Bearer
2. **Hash de contraseñas con bcrypt:**
    - Salt rounds: 10 (balance entre seguridad y rendimiento)
    - Hash automático en pre-save hook
3. **Validación exhaustiva de inputs:**

    - Validación de email con librería `validator`
    - Límites de caracteres en campos de texto
    - Verificación de tipos de datos
    - Sanitización de inputs (trim, toLowerCase)
4. **CORS configurado correctamente:**
    - Whitelist para desarrollo y producción
5. **Prevención de NoSQL injection:**
    - Validación de ObjectId antes de queries
    - Uso de queries parametrizadas de Mongoose

Validación de Datos

1. **Validador estructural de diagramas:**
    - Archivo `validators/diagramValidator.js`
    - Validación de nodos, edges e imágenes
    - Validación de referencias entre edges y nodes
    - Validación de metadatos de imágenes (tamaño, tipo MIME)
2. **Validación a nivel de modelo:**
    - Validaciones declarativas en esquemas
3. **Validación a nivel de controlador:**
    - Validación manual de lógica de negocio
    - Mensajes de error descriptivos y específicos
    - Códigos HTTP apropiados (400, 401, 404, 409, 500)

Rendimiento y Optimización

1. **Queries optimizadas:**
    - Select: solo campos necesarios
    - Sort: ordenamiento en DB
    - Lean: devuelve POJO en lugar de documentos Mongoose
2. **Límites de tamaño:**
    - JSON body limit: 10MB (para imágenes base64)
    - Imágenes: máximo 5MB por imagen
    - Máximo 10 imágenes por diagrama
    - Máximo 10 juegos favoritos por usuario
3. **Índices únicos en MongoDB:**

- Índices en campos email para búsquedas rápidas

Código Limpio y Mantenible

1. **Documentación JSDoc completa:**
   - Todos los endpoints documentados
   - Parámetros y respuestas especificados
   - Tipos de datos claramente indicados
2. **Separación de responsabilidades:**
   - Config, Models, Controllers, Routes, Middleware, Validators en carpetas separadas
3. **Nombres descriptivos:**
   - Variables, funciones y rutas con nombres claros
   - Convenciones de nomenclatura consistentes
4. **Constantes extraídas:**
   - Valores mágicos evitados
   - Definición clara de límites y configuraciones

2.3. Integración de Librerías Externas

Backend

Librería	Versión	Justificación
express	^5.1.0	Framework web minimalista y maduro
mongoose	^8.19.3	ODM para MongoDB con validación integrada
bcrypt	^6.0.0	Hashing seguro de contraseñas
jsonwebtoken	^9.0.2	Implementación JWT estándar
validator	^13.15.23	Validación y sanitización de strings
cors	^2.8.5	Middleware CORS configurable
dotenv	^17.2.3	Carga de variables de entorno
nodemailer	^6.9.7	Envío de emails transaccionales
nodemon	^3.1.10	Hot reload en desarrollo

Frontend

Librería	Versión	Justificación
react	^18.3.1	Biblioteca UI declarativa con hooks
react-dom	^18.3.1	Renderizado en el DOM
react-router-dom	^6.26.0	Routing del lado del cliente
reactflow	^11.11.4	Editor visual de diagramas
axios	^1.7.7	Cliente HTTP con interceptors
react-icons	^5.5.0	Iconografía tree-shakeable
html-to-image	^1.11.11	Exportación a PNG/JPG
jspdf	^3.0.4	Generación de PDFs
react-toastify	^10.0.5	Notificaciones toast
vite	^5.4.21	Build tool moderno
eslint	^8.57.1	Linter JavaScript
prettier	^3.3.3	Formateador de código

3. Completitud del Proyecto

3.1. Funcionalidades Principales Implementadas

Sistema de Autenticación Completo

- [x] Registro de usuarios con validación
- [x] Login con generación de JWT
- [x] Logout
- [x] Opción "Remember Me"
- [x] Middleware de autenticación
- [x] Protección de rutas privadas

CRUD de Diagramas

- [x] Crear diagrama
- [x] Leer diagramas
- [x] Actualizar diagrama
- [x] Eliminar diagrama

- [x] Sistema de plantillas

Editor Visual de Diagramas (React Flow)

- [x] Creación de nodos
- [x] Conexión de nodos
- [x] Drag & drop
- [x] Zoom y pan
- [x] Edición de propiedades
- [x] Eliminación de elementos
- [x] Guardado persistente

Sistema de Imágenes

- [x] Upload de imágenes base64
- [x] Validación de URLs
- [x] Almacenamiento en servidor
- [x] Metadatos completos
- [x] Asociación a nodos
- [x] Eliminación de imágenes

Gestión de Perfiles

- [x] Visualización de perfil
- [x] Actualización de datos
- [x] Sistema de logros
- [x] Estadísticas

Importación/Exportación

- [x] Exportación JSON
- [x] Exportación PNG/JPG
- [x] Exportación PDF
- [x] Importación desde JSON

3.2. Nivel de Finalización

Estimación: 95% completo

El proyecto cumple con todos los requisitos del MVP y las ampliaciones planificadas.

3.3. Validación Multicapa

1. **Frontend**: Validación básica de formularios
2. **Backend - Routes**: Middleware de autenticación

3. **Backend - Controllers**: Validación de lógica de negocio
4. **Backend - Validators**: Validación estructural compleja
5. **Backend - Models**: Validación de esquema Mongoose

4. Innovación y Creatividad

4.1. Originalidad de la Idea

BossFlow es un concepto único que combina gaming + productividad. Permite a gamers documentar y compartir estrategias de videojuegos en formato visual.

Diferenciación:

- Miro, Lucidchart, Draw.io: Generalistas
- Wikis de juegos: Texto e imágenes estáticas
- Foros: Difícil de buscar y reutilizar

4.2. Características Únicas

Tipos de Nodos Especializados:

- actionNode: Acciones del jugador
- decisionNode: Puntos de decisión
- bossNode: Representación del jefe
- conditionNode: Condiciones del juego
- timerNode: Eventos temporales
- effectNode: Efectos y buffs

Sistema de Imágenes Integrado:

- Upload base64 hasta 5MB
- URLs externas válidas
- Asociación a nodos
- Máximo 10 imágenes por diagrama

Plantillas Reutilizables:

- Flag `isTemplate` para marcar plantillas
- Endpoint separado para templates
- Los usuarios pueden crear y compartir estrategias base

Importación/Exportación Múltiple:

- Formatos: JSON, PNG/JPG, PDF
- Facilita compartir en redes sociales y wikis

4.3. Valor Añadido

Para Jugadores Individuales:

- Documentar estrategias personales
- No olvidar tácticas complejas
- Mejorar progresivamente

Para Speedrunners:

- Documentar rutas óptimas
- Visualizar splits y decisiones
- Compartir con la comunidad

Para Streamers:

- Explicar estrategias en streams
- Crear contenido educativo
- Compartir en redes sociales

Para Guilds y Equipos:

- Coordinación de raids
- Estrategias de equipo
- Onboarding de miembros

5 Configuracionesn Inicial

5.1. Instalación y ejecución

Requisitos previos:

- Node.js 18+ y npm (solo para desarrollo local).
- Docker y docker-compose (recomendado para despliegue o para levantar todo el stack fácilmente).

1) Clonar el repositorio

git clone https://github.com/GunterMagno/BossFlow.git

cd BossFlow

2.1 Desarrollo local (sin Docker)

- Backend:

cd backend

npm install

npm run dev

- Frontend:

cd frontend

npm install

npm run dev

2.2 Levantar con Docker Compose (modo desarrollo)

docker compose -f docker-compose.dev.yml up --build

2.3 Levantar con Docker Compose (modo producción)

docker compose -f docker-compose.prod.yml up --build -d

Variables de entorno (ejemplos)

Backend (archivo `.env` en `backend/`):

MONGO_URI=mongodb://mongo:27017/bossflow

JWT_SECRET=tu_secreto_jwt

PORT=4000

Frontend (archivo `.env` en `frontend/` o en tu entorno):

VITE_API_URL=http://localhost:4000/api

5.2. Guía de usuario

6. Metodología y organización SCRUM

Equipo del Proyecto

- **Jesús López Pérez**: Product Owner / Frontend Lead / DevOps
- **Alejandro Borrego**: Backend Developer / QA / Testing
- **Daniel Montes**: Scrum Master / Full-stack Developer / UI&UX Designer

Resumen de Sprints

Sprint	Fechas	Jesús	Alejandro	Daniel
Sprint 1	31 Oct - 6 Nov 2024	Product Owner	Scrum Master	Developer
Sprint 2	7 Nov - 13 Nov 2024	Scrum Master	Developer	Product Owner
Sprint 3	14 Nov - 20 Nov 2024	Developer	Product Owner	Scrum Master
Sprint 4	21 Nov - 27 Nov 2024	Product Owner	Scrum Master	Developer
Sprint 5	28 Nov - 4 Dic 2024	Scrum Master	Developer	Product Owner
Sprint 6	5 Dic - 12 Dic 2024	Developer	Product Owner	Scrum Master

SPRINT 1: Setup y Arquitectura Base

Duración: Semana 1 (31 Oct - 6 Nov 2024)

Estimación Total: 58 horas

Issues: 20

Objetivo del Sprint

Establecer la base técnica del proyecto configurando el repositorio, las herramientas de desarrollo, y la estructura inicial tanto del backend (Node.js + Express + MongoDB) como del frontend (React + Vite). Configurar la infraestructura completa incluyendo estructura de carpetas, dependencias, base de datos y entorno de desarrollo. El equipo también debe familiarizarse con las tecnologías clave del stack MERN y establecer los flujos de trabajo colaborativo con Git/GitHub.

Issues

#	Tarea	Responsable	Estimación (h)
#1	Aprender fundamentos de Node.js	Alejandro Borrego	5
#2	Aprender Express.js básico	Alejandro Borrego	4
#3	Setup proyecto	Alejandro Borrego	3

	Backend		
#4	Crear ruta GET /api/health	Alejandro Borrego	2
#5	Configurar MongoDB local	Alejandro Borrego	3
#6	Facilitar Daily Scrums	Alejandro Borrego	2
#7	Documentar setup Backend	Alejandro Borrego	1
#8	Aprender fundamentos de React	Jesús López Pérez	6
#9	Aprender React Hooks	Jesús López Pérez	3
#10	Setup proyecto Frontend	Jesús López Pérez	3
#11	Crear componente Home básico	Jesús López Pérez	2
#12	Conectar Frontend con Backend	Jesús López Pérez	3
#13	Crear Product Backlog inicial	Jesús López Pérez	2
#14	Documentar setup Frontend	Jesús López Pérez	1
#15	Aprender Git/GitHub workflow	Daniel Montes	2
#16	Configurar repositorio GitHub	Daniel Montes	2
#17	Aprender MongoDB + Mongoose	Daniel Montes	5
#18	Investigar API Rawg.io	Daniel Montes	3
#19	Diseñar esquema DB inicial	Daniel Montes	3
#20	Setup entorno de desarrollo	Daniel Montes	3

SPRINT 2: Autenticación y Gestión de Usuarios

Duración: Semana 2 (7 Nov - 13 Nov 2024)

Estimación Total: 60 horas

Issues: 21

Objetivo del Sprint

Implementar un sistema de autenticación completo y seguro utilizando JWT que permita a los usuarios registrarse, iniciar sesión y mantener su sesión activa. Desarrollar el sistema completo de autenticación con registro, login y protección de rutas. Tanto el backend como el

frontend deben proteger las rutas que requieren autenticación, y se debe diseñar la interfaz de usuario básica con wireframes y un sistema de diseño coherente para las siguientes fases del proyecto.

Issues

#	Tarea	Responsable	Estimación (h)
#21	Crear modelo User en MongoDB	Alejandro Borrego	3
#22	Implementar registro de usuarios	Alejandro Borrego	4
#23	Implementar login de usuarios	Alejandro Borrego	4
#24	Crear middleware de autenticación	Alejandro Borrego	3
#25	Implementar logout	Alejandro Borrego	2
#26	Testing con Insomnia	Alejandro Borrego	3
#27	Code review	Alejandro Borrego	1
#28	Crear componente de Login	Jesús López Pérez	4
#29	Crear componente de Registro	Jesús López Pérez	4
#30	Implementar Context API para Auth	Jesús López Pérez	4
#31	Conectar Login/Registro con Backend	Jesús López Pérez	3
#32	Guardar JWT en localStorage	Jesús López Pérez	2
#33	Facilitar ceremonias SCRUM	Jesús López Pérez	2
#34	Documentar flujo de autenticación	Jesús López Pérez	1
#35	Diseñar wireframes básicos	Daniel Montes	4
#36	Crear sistema de diseño	Daniel Montes	3
#37	Implementar navbar básico	Daniel Montes	3
#38	Crear layout principal	Daniel Montes	3
#39	Implementar rutas protegidas	Daniel Montes	3
#40	Refinar historias Sprint 3	Daniel Montes	2
#41	Validar MVP Sprint 2	Daniel Montes	2

SPRINT 3: CRUD de Diagramas

Duración: Semana 3 (14 Nov - 20 Nov 2024)

Estimación Total: 55 horas

Issues: 21

Objetivo del Sprint

Desarrollar la funcionalidad central del proyecto implementando el CRUD completo de diagramas, permitiendo a los usuarios crear, listar, visualizar y eliminar sus diagramas desde un dashboard. Implementar las operaciones de crear, leer, actualizar y eliminar diagramas con su dashboard correspondiente. Se integra React Flow como base del editor visual y se establecen las rutas y componentes necesarios para la navegación entre el dashboard y el editor de diagramas.

Issues

#	Tarea	Responsable	Estimación (h)
#42	Definir criterios de aceptación CRUD	Alejandro Borrego	2
#43	Crear modelo Diagram	Alejandro Borrego	3
#44	Implementar POST /api/diagrams	Alejandro Borrego	3
#45	Implementar GET /api/diagrams	Alejandro Borrego	3
#46	Implementar DELETE /api/diagrams/:id	Alejandro Borrego	3
#47	Implementar GET /api/diagrams/:id	Alejandro Borrego	2
#48	Testing endpoints con Insomnia	Alejandro Borrego	2
#49	Crear página Dashboard	Jesús López Pérez	2
#50	Implementar listado de diagramas	Jesús López Pérez	4
#51	Crear modal para nuevo diagrama	Jesús López Pérez	3
#52	Conectar creación con backend	Jesús López Pérez	3
#53	Implementar eliminación de diagramas	Jesús López Pérez	3
#54	Añadir mensajes de feedback	Jesús López Pérez	2
#55	Crear componente DiagramCard	Jesús López Pérez	2
#56	Instalar y configurar	Daniel Montes	2

	React Flow		
#57	Crear página Editor básica	Daniel Montes	3
#58	Implementar React Flow básico	Daniel Montes	4
#59	Crear ruta desde Dashboard a Editor	Daniel Montes	2
#60	Implementar estado inicial vacío	Daniel Montes	2
#61	Organizar Daily Scrums	Daniel Montes	3
#62	Documentar componente Editor	Daniel Montes	2

SPRINT 4: Editor Visual y Funcionalidades Avanzadas

Duración: Semana 4 (21 Nov - 27 Nov 2024)

Estimación Total: 62 horas

Issues: 21

Objetivo del Sprint

Implementar el editor visual completo con React Flow incluyendo nodos personalizados, conexiones, drag & drop, guardado y carga. Desarrollar nodos personalizados para diferentes tipos de elementos, permitir la creación de conexiones entre nodos, implementar la funcionalidad de guardado y carga de diagramas desde el servidor, y añadir capacidades de drag & drop para una experiencia de usuario fluida.

Issues

#	Tarea	Responsable	Estimación (h)
#63	Actualizar modelo Diagram con nodes	Alejandro Borrego	2
#64	Implementar PUT /api/diagrams/:id	Alejandro Borrego	3
#65	Optimizar queries MongoDB	Alejandro Borrego	3
#66	Implementar validaciones backend	Alejandro Borrego	3
#67	Añadir timestamps	Alejandro Borrego	2
#68	Testing de guardado/carga	Alejandro Borrego	3
#69	Documentar estructura de datos	Alejandro Borrego	2
#70	Definir tipos de nodos	Jesús López Pérez	4
#71	Implementar panel lateral	Jesús López Pérez	3
#72	Implementar drag &	Jesús López Pérez	4

	drop de nodos		
#73	Crear modal de edición de nodos	Jesús López Pérez	4
#74	Implementar eliminación de nodos	Jesús López Pérez	2
#75	Añadir toolbar al editor	Jesús López Pérez	3
#76	Conectar guardado con backend	Jesús López Pérez	4
#77	Implementar conexión de nodos	Daniel Montes	4
#78	Crear custom nodes personalizados	Daniel Montes	5
#79	Implementar guardado en backend	Daniel Montes	3
#80	Implementar carga de diagrama	Daniel Montes	4
#81	Mejorar UI del editor	Daniel Montes	2
#82	Mejorar UX del editor	Daniel Montes	1
#83	Code review y testing	Daniel Montes	1

SPRINT 5: Funcionalidades Avanzadas y Testing

Duración: Semana 5 (28 Nov - 4 Dic 2024)

Estimación Total: 53 horas

Issues: 19

Objetivo del Sprint

Enriquecer la aplicación con funcionalidades avanzadas como soporte para imágenes en nodos, sistema de plantillas predefinidas, capacidades de exportación e importación en múltiples formatos (JSON, PDF, PNG), y realizar testing exhaustivo del MVP. Agregar características avanzadas incluyendo soporte de imágenes, templates, exportación/importación, testing exhaustivo y despliegue en producción.

Issues

#	Tarea	Responsable	Estimación (h)
#84	Backend: actualizar modelo para imágenes	Alejandro Borrego	3
#85	Frontend: UI para gestión de imágenes	Alejandro Borrego	3
#86	Backend: endpoints para subida de	Alejandro Borrego	2

	imágenes		
#87	Frontend: renderizado de imágenes en editor	Alejandro Borrego	2
#88	Frontend: popup con descripción en nodo	Alejandro Borrego	1.5
#89	Frontend: popup/hover de descripción (UX)	Alejandro Borrego	1
#177	Tests y documentación: imágenes	Alejandro Borrego	1.5
#90	Build de producción Frontend	Jesús López Pérez	2
#91	Desplegar Frontend	Jesús López Pérez	2
#92	Testing completo del MVP	Jesús López Pérez	4
#93	Crear presentación final	Jesús López Pérez	5
#94	Documentación de usuario	Jesús López Pérez	3
#95	Testing de aceptación final	Jesús López Pérez	2
#96	Facilitar ceremonias finales	Jesús López Pérez	2
#181	Crear página Plantillas	Daniel Montes	4
#182	Implementar plantillas de diagramas	Daniel Montes	3
#184	Implementar exportación de diagramas	Daniel Montes	4
#185	Implementar importación/exportación JSON	Daniel Montes	4
#189	Arreglar e implementar funcionalidades	Daniel Montes	2

SPRINT 6: Documentación y Cierre del Proyecto

Duración: Semana 6 (5 Dic - 12 Dic 2024)

Estimación Total: 29 horas

Issues: 10

Objetivo del Sprint

Cerrar el proyecto completando toda la documentación técnica y de usuario necesaria para la entrega final. Completar la documentación final, video demo, presentación y entrega del proyecto. Esto incluye la documentación de la Fase 3, la creación de un video demostrativo, la actualización del README principal, la documentación de arquitectura técnica, y la preparación de la presentación final.

Issues

#	Tarea	Responsable	Estimación (h)
#97	Documentar Fase 3 del proyecto	Daniel Montes	8
#98	Corregir bugs críticos finales	Daniel Montes	2
#99	Crear video demo	Daniel Montes	3
#100	Actualizar README principal	Daniel Montes	2
#101	Documentar arquitectura técnica	Daniel Montes	2
#102	Retrospectiva final del proyecto	Daniel Montes	2
#103	Preparar entrega final	Daniel Montes	2
-	Trabajar en documentación	Jesús López Pérez	4
-	Trabajar en documentación	Alejandro Borrego	4