Sprint 2 Practico 3 Final

Estructura Completa de Carpetas

```
📂 src
|
├── 📁 config                      # ⚙️ Configuración de la base de datos
|   └── 📄 dbConfig.mjs            # 🛠️ Conexión a MongoDB
|
├── 📁 models                      # 📄 Modelos de datos
|   └── 📄 SuperHero.mjs           # 🦸 Modelo SuperHero
|
├── 📁 repositories                # 💾 Capa de persistencia
|   ├── 📄 IRepository.mjs         # 📜 Interfaz CRUD
|   └── 📄 SuperHeroRepository.mjs # 🔄 Implementación CRUD de SuperHero
|
├── 📁 services                    # 🧠 Lógica de negocio
|   └── 📄 superheroesService.mjs  # 🛠️ Servicios de SuperHero
|
├── 📁 controllers                 # 🌐 Controladores
|   └── 📄 superheroesController.mjs # 🕹️ Controlador para superhéroes
|
├── 📁 routes                      # 🖼️ Rutas de la API
|   └── 📄 superHeroRoutes.mjs     # 🌄 Rutas API para superhéroes
|
├── 📁 views                       # 👀 Capa de presentación
|   └── 📄 responseView.mjs        # 🌐 Funciones para mostrar datos de superhéroes
|
└── 📄 app.mjs                     # 🚀 Archivo principal de la aplicación
```

Paso 1: Configuración de la Conexión a MongoDB en `dbConfig.mjs`

Archivo: `src/config/dbConfig.mjs`

Funcionalidad: Este archivo configura la conexión centralizada a MongoDB, permitiendo que la aplicación tenga una única instancia de conexión que puede ser utilizada en cualquier parte del proyecto.

```
import mongoose from 'mongoose';

export async function connectDB() {
    try {
        await mongoose.connect('mongodb+srv://Grupo-XX:grupoXX@cursadanodejs.
            ls9ii.mongodb.net/Node-js');
        console.log('Conexión exitosa a MongoDB');
    } catch (error) {
        console.error('Error al conectar a MongoDB:', error);
        process.exit(1);
    }
}
```

Justificación teórica: Centralizar la configuración de conexión a MongoDB en `dbConfig.mjs` permite tener un único punto de configuración. Esto facilita el mantenimiento y asegura que cualquier cambio en la configuración se realice en un solo lugar, mejorando la modularidad y la reusabilidad del código.

Paso 2: Definir el Modelo de Datos SuperHero en `SuperHero.mjs`

Archivo: `src/models/SuperHero.mjs`

Funcionalidad: Define el modelo de datos para superhéroes utilizando Mongoose, estableciendo la estructura y las reglas de validación para los documentos que serán almacenados en MongoDB.

```
import mongoose from 'mongoose';

const superheroSchema = new mongoose.Schema({
    nombreSuperHeroe: { type: String, required: true },
    nombreReal: { type: String, required: true },
    edad: { type: Number, min: 0 },
    planetaOrigen: { type: String, default: 'Desconocido' },
    debilidad: String,
    poderes: [String],
    aliados: [String],
    enemigos: [String],
    creador: String,
    createdAt: { type: Date, default: Date.now }
});
const superHero =  mongoose.model('SuperHero', superheroSchema, 'Grupo-XX');
export default superHero
```

Justificación teórica: Definir el modelo de datos asegura que cada documento en la colección de superhéroes siga una estructura consistente, lo que permite tener un control de calidad sobre los datos. Mongoose facilita la validación y la gestión de los datos, garantizando que cada documento cumpla con los requisitos del esquema, como la obligatoriedad de ciertos campos y los tipos de datos.

Paso 3: Crear la Capa de Persistencia `SuperHeroRepository.mjs` con la Interfaz `IRepository.mjs`

Archivo de Interfaz `IRepository.mjs`

Funcionalidad: `IRepository.mjs` establece una interfaz que define métodos CRUD estándar y sirve como contrato para asegurar que cualquier clase que implemente la interfaz cuente con estos métodos.

```
class IRepository {
    obtenerPorId(id) {
        throw new Error("Método 'obtenerPorId()' no implementado");
    }
    obtenerTodos() {
        throw new Error("Método 'obtenerTodos()' no implementado");
    }
    buscarPorAtributo(atributo, valor) {
        throw new Error("Método 'buscarPorAtributo()' no implementado");
    }
```

```
    obtenerMayoresDe30() {
        throw new Error("Método 'obtenerMayoresDe30()' no implementado");
    }
}

export default IRepository;
```

Justificación teórica: La interfaz `IRepository.mjs` proporciona una abstracción de los métodos CRUD que deben ser implementados por cualquier repositorio. Esto asegura que todas las clases de repositorio mantengan consistencia en sus métodos, mejorando la mantenibilidad y facilitando cambios futuros en la implementación.

Archivo de Implementación `SuperHeroRepository.mjs`

Archivo: `src/repositories/SuperHeroRepository.mjs`

```
import SuperHero from '../models/SuperHero.mjs';
import IRepository from './IRepository.mjs';

class SuperHeroRepository extends IRepository {
    async obtenerPorId(id) {
        return await SuperHero.findById(id);
    }

    async obtenerTodos() {
        return await SuperHero.find({});
    }

    async buscarPorAtributo(atributo, valor) {
        RESOLVER
    }

    async obtenerMayoresDe30() {
        RESOLVER
    }
}

export default new SuperHeroRepository();
```

Justificación teórica: `SuperHeroRepository.mjs` implementa los métodos definidos en la interfaz, interactuando directamente con MongoDB a través de Mongoose para realizar operaciones de datos. La centralización de estas operaciones en el repositorio mejora la organización y garantiza que el acceso a los datos se realice de manera controlada y uniforme.

Paso 4: Crear la Capa de Servicio `superheroesService.mjs`

Archivo: `src/services/superheroesService.mjs`

Funcionalidad: Este archivo implementa la lógica de negocio, utilizando los métodos del repositorio para recuperar, buscar y filtrar datos de los superhéroes.

```
import superHeroRepository from '../repositories/SuperHeroRepository.mjs';
```

```
export async function obtenerSuperheroePorId(id) {
    return await superHeroRepository.obtenerPorId(id);
}

export async function obtenerTodosLosSuperheroes() {
    return await superHeroRepository.obtenerTodos();
}

export async function buscarSuperheroesPorAtributo(atributo, valor) {
    return await superHeroRepository.buscarPorAtributo(atributo, valor);
}

export async function obtenerSuperheroesMayoresDe30() {
    return await superHeroRepository.obtenerMayoresDe30();
}
```

Justificación teórica: La capa de servicios contiene la lógica de negocio y se encarga de validar y transformar los datos cuando es necesario. Esto permite mantener el repositorio enfocado únicamente en el acceso a la base de datos, facilitando la separación de responsabilidades y asegurando que la lógica de negocio se encuentre en un solo lugar.

Paso 5: Crear el Controlador `superheroesController.mjs`

Archivo: `src/controllers/superheroesController.mjs`

Funcionalidad: Este archivo implementa el controlador para gestionar las solicitudes HTTP, llamando a los servicios correspondientes y utilizando las vistas para presentar los datos.

```
import { obtenerSuperheroePorId, obtenerTodosLosSuperheroes,
 buscarSuperheroesPorAtributo, obtenerSuperheroesMayoresDe30 }
from '../services/superheroesService.mjs';
import { renderizarSuperheroe, renderizarListaSuperheroes }
 from '../views/responseView.mjs';

export async function obtenerSuperheroePorIdController(req, res) {
 try {
        const { id } = req.params;
        const superheroe = await obtenerSuperheroePorId(id);
        if (!superheroe) {
            return res.status(404).send({ mensaje: 'Superhéroe no encontrado' });
        }

        const superheroeFormateado = renderizarSuperheroe(superheroe);
        res.status(200).json(superheroeFormateado);
    } catch (error) {
        res.status(500).send({ mensaje: 'Error al obtener el superhéroe',
         error: error.message });
    }
}

export async function obtenerTodosLosSuperheroesController(req, res) {
      try {
```

```
        const superheroes = await obtenerTodosLosSuperheroes();

        const superheroesFormateados = renderizarListaSuperheroes(superheroes);
        res.status(200).json(superheroesFormateados);
    } catch (error) {
        res.status(500).send({ mensaje: 'Error al obtener los superhéroes',
         error: error.message });
    }
}

export async function buscarSuperheroesPorAtributoController(req, res) {
try {
        const { atributo, valor } = req.params;
        const superheroes = await buscarSuperheroesPorAtributo(atributo, valor);
        if (superheroes.length === 0) {
            return res.status(404).send(
                { mensaje: 'No se encontraron superhéroes con ese atributo' });
        }

        const superheroesFormateados = renderizarListaSuperheroes(superheroes);
        res.status(200).json(superheroesFormateados);
    } catch (error) {
        res.status(500).send({ mensaje: 'Error al buscar los superhéroes',
            error: error.message });
    }
}

export async function obtenerSuperheroesMayoresDe30Controller(req, res) {
    try {
        const superheroes = await obtenerSuperheroesMayoresDe30();
        if (superheroes.length === 0) {
            return res.status(404).send(
                { mensaje: 'No se encontraron superhéroes mayores de 30 años' });
        }
        const superheroesFormateados = renderizarListaSuperheroes(superheroes);
        res.status(200).json(superheroesFormateados);
    } catch (error) {
        res.status(500).send(
        {mensaje: 'Error al obtener superhéroes mayores de 30', error: error.message});
    }
}
```

Justificación teórica: La capa de controladores gestiona las solicitudes del cliente y llama a la capa de servicios para realizar las operaciones necesarias. Al usar funciones específicas para cada endpoint, el controlador actúa como intermediario, facilitando la separación de responsabilidades y mejorando la organización del código.

Paso 6: Configurar la Capa de Vistas en `responseView.mjs`

Archivo: `src/views/responseView.mjs`

Funcionalidad: Este archivo define las funciones de presentación de los datos, organizando la información de los superhéroes en un formato estructurado.

```
export function renderizarSuperheroe(superheroe) {
    return {
        Nombre: superheroe.nombreSuperHeroe,
        "Nombre Real": superheroe.nombreReal,
        Edad: superheroe.edad,
        "Planeta de Origen": superheroe.planetaOrigen,
        Debilidad: superheroe.debilidad,
        Poderes: superheroe.poderes,
        Aliados: superheroe.aliados,
        Enemigos: superheroe.enemigos
    };
}

export function renderizarListaSuperheroes(superheroes) {
    return superheroes.map(superheroe => renderizarSuperheroe(superheroe));
}
```

Justificación teórica: La capa de vistas organiza la presentación de los datos, facilitando la lectura y el acceso a la información en un formato estándar para el cliente. Esto mejora la separación de responsabilidades y permite mantener la lógica de presentación en un único lugar.

Paso 7: Configuración de las Rutas en `superHeroRoutes.mjs`

Archivo: `src/routes/superHeroRoutes.mjs`

Funcionalidad: Este archivo define las rutas necesarias para cada operación del controlador.

```
import express from 'express';
import {
    obtenerSuperheroePorIdController,
    obtenerTodosLosSuperheroesController,
    buscarSuperheroesPorAtributoController,
    obtenerSuperheroesMayoresDe30Controller
} from '../controllers/superheroesController.mjs';

const router = express.Router();

router.get('/heroes', obtenerTodosLosSuperheroesController);
router.get('/heroes/:id', obtenerSuperheroePorIdController);
router.get('/heroes/buscar/:atributo/:valor', buscarSuperheroesPorAtributoController);
router.get('/heroes/mayores-30', obtenerSuperheroesMayoresDe30Controller);

export default router;
```

Justificación teórica: La capa de rutas define los endpoints y mapea cada uno a su respectivo controlador, permitiendo que las solicitudes HTTP se manejen de forma estructurada y predecible.

Paso Final: Configuración del Servidor en `app.mjs`

Archivo: `src/app.mjs`

Funcionalidad: Inicializa el servidor, conecta a la base de datos, y carga las rutas para gestionar todas las solicitudes relacionadas con superhéroes.

```
import express from 'express';
import { connectDB } from './config/dbConfig.mjs';
import superHeroRoutes from './routes/superHeroRoutes.mjs';

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware para parsear JSON
app.use(express.json());

// Conexión a MongoDB
connectDB();

// Configuración de rutas
app.use('/api', superHeroRoutes);

// Manejo de errores para rutas no encontradas
app.use((req, res) => {
    res.status(404).send({ mensaje: "Ruta no encontrada" });
});

// Iniciar el servidor
app.listen(PORT, () => {
    console.log(`Servidor escuchando en el puerto ${PORT}`);
});
```

Justificación teórica: `app.mjs` centraliza la configuración de la aplicación, conectando a MongoDB y cargando las rutas necesarias. Esto permite una configuración modular y fácilmente escalable, asegurando que la aplicación esté lista para manejar solicitudes en el entorno de desarrollo o producción.