Afondamento nas Competencias Profesionais

Actividad 1.3 - Arquitectura

Documentación de código: Express API

Daniel Calvar Cruz, 2ºCS DAM

Índice

1. Descripción general

Volver

Es una API basada en el modelo estándar de aplicaciones web, derivado del clásico Vista-Controlador.

Al ser una API pequeña tiene un funcionamiento muy simple, se ocupa de llevar a cabo acciones CRUD con una base de datos MongoDB y devolver al frontend los datos pertinentes.

2. Información técnica y dependencias

Volver

- **Versión Node.js:** 22.14
- **Base de datos:** MongoDB Atlas
- **Dependencias NPM:**
  - **cors:** 2.8.5
  - **dotenv:** 16.4.7
  - **express:** 4.21.1
  - **mongodb:** 6.12.0
  - **mongoose:** 8.13.2
  - **nodemon:** 3.1.10

3. Configuración

Volver

3.1. Variables de entorno (.env)

- `MONGO_URI`: URI de conexión a MongoDB Atlas o local.
- `PORT`: Puerto en el que se ejecutará el servidor.
- `DDBB_NAME`: Nombre de la base de datos en Mongo Atlas.

3.2. Archivo vercel.json

Define la configuración para el despliegue en Vercel, incluyendo las rutas y la carpeta de salida de la API.

3.3. CORS

El middleware `cors.js` define los dominios permitidos para acceder a la API, garantizando la seguridad y evitando errores de origen cruzado.

4. Estructura

Volver

```
├── api/
│   ├── config/
│   │   └── ddbb.js
│   ├── controllers/
│   │   └── crudController.js
│   ├── middlewares/
│   │   └── cors.js
│   ├── models/
│   │   ├── Module.js
│   │   └── index.js
│   ├── routes/
│   │   ├── crud.js
│   │   └── index.js
│   ├── services/
│   │   └── crudServices.js
│   ├── app.js
│   └── server.js
├── node_modules
├── package-lock.json
├── package.json
└── vercel.json
```

- (root) node-modules: paquetes de node.
- (root) package-lock.json / package.json: archivos de configuración.
- (root) vercel.json: configuración de despliegue.
- (api) config: conexión a la base de datos.
- (api) middlewares: middlewares del sistema.
- (api) models: esquemas de Mongoose.
- (api) routes: rutas de entrada de los endpoints.
- (api) controllers: enlaza las rutas de entrada con los servicios.
- (api) services: lógica de negocio.

- (api root) app: configuración del servidor Express.
- (api root) server: punto de entrada y conexión del servidor.

5. Modelo de datos

Volver

El modelo principal se define en `models/Module.js` y representa una ficha de personaje.

Campo	Tipo	Descripción
nombre	String	Nombre del personaje
clase	String	Clase o rol del personaje
HP	Number	Puntos de vida
indice	Number	Índice interno autogenerado

Ejemplo de documento JSON:

```
{
  "nombre": "Mustakrakish",
  "clase": "Trol del lago",
  "HP": 9999,
  "indice": 1
}
```

6. Arquitectura

Volver

La API sigue una arquitectura modular basada en el patrón MVC (Modelo–Vista–Controlador).

- **Rutas:** reciben la petición HTTP y determinan qué controlador usar.
- **Controladores:** validan y gestionan las peticiones, llamando a los servicios necesarios.
- **Servicios:** contienen la lógica de negocio y se comunican con la base de datos.
- **Modelos:** definen la estructura de los datos en MongoDB mediante Mongoose.

Flujo de ejecución de una petición:

```
Cliente → Ruta → Controlador → Servicio → Modelo (MongoDB) → Respuesta
```

7. API

Volver

7.1. Endpoints detallados

7.1.1. /create

Crea una nueva ficha en la base de datos. Crea el índice automáticamente.

Tipo: **POST**

Body:

Parámetro	Tipo	Descripción
nombre	String	Nombre del personaje de la ficha
clase	String	Clase del personaje de la ficha
HP	int	Puntos de vida del personaje

Respuesta:

Parámetro	Tipo	Descripción
type	String	success / failure
message	String	Mensaje de éxito o fallo

Ejemplo completo de uso:

```
POST /create
{
  "nombre": "Ragnar",
  "clase": "Guerrero",
  "HP": 100
}

Respuesta:
{
  "type": "success",
  "message": "Ficha creada correctamente"
}
```

7.1.2. /read

Devuelve un documento en concreto.

Tipo: **POST**

Body:

Parámetro	Tipo	Descripción
filtroKey	String	Clave a buscar en base de datos
filtroValue	String	Valor que va a buscar usando la clave

Respuesta:

Parámetro	Tipo	Descripción
type	String	success / failure
message	String	Mensaje de éxito o fallo
data	Objeto JSON	Datos del documento encontrado

Ejemplo completo de uso:

```
POST /read
{
  "filtroKey": "nombre",
  "filtroValue": "Ragnar"
}

Respuesta:
{
  "type": "success",
  "message": "Documento encontrado",
  "data": {
    "nombre": "Ragnar",
    "clase": "Guerrero",
    "HP": 100,
    "indice": 1
  }
}
```

7.1.3. /update

Modifica un documento en concreto.

Tipo: **POST**

Body:

Parámetro	Tipo	Descripción
indice	int	Índice interno en base de datos de la ficha
nombre	String	Nombre del personaje
clase	String	Clase del personaje
HP	int	Puntos de vida del personaje

Respuesta:

Parámetro	Tipo	Descripción
type	String	success / failure
message	String	Mensaje de éxito o fallo

Ejemplo completo de uso:

```
POST /update
{
  "indice": 1,
  "nombre": "Ragnar",
  "clase": "Berserker",
  "HP": 130
}

Respuesta:
{
  "type": "success",
  "message": "Ficha actualizada correctamente"
}
```

7.1.4. /delete

Borra un documento en concreto.

Tipo: **POST**

Body:

Parámetro	Tipo	Descripción
indice	int	Índice interno en base de datos de la ficha

Respuesta:

Parámetro	Tipo	Descripción
type	String	success / failure
message	String	Mensaje de éxito o fallo

Ejemplo completo de uso:

```
POST /delete
{
  "indice": 1
}

Respuesta:
{
  "type": "success",
  "message": "Ficha eliminada correctamente"
}
```

7.1.5. /readall

Devuelve todos los documentos de la base de datos.

Tipo: **GET**

Respuesta:

Parámetro	Tipo	Descripción
type	String	success / failure
message	String	Mensaje de éxito o fallo
data	Array de objetos JSON	Lista de todos los documentos encontrados

Ejemplo completo de uso:

```
GET /readall
```

Respuesta:


```
{
  "type": "success",
  "message": "Documentos encontrados",
  "data": [
    {
      "nombre": "Ragnar",
      "clase": "Berserker",
      "HP": 130,
      "indice": 1
    },
    {
      "nombre": "Luna",
      "clase": "Maga",
      "HP": 80,
      "indice": 2
    }
  ]
}
```

8. Pruebas

Volver

- Para probar la API localmente:

```
npm run dev
```

- Recomendado usar herramientas como **Postman** o la extensión de VS Code **Thunderclient**.
- Ejemplo:

```
POST http://localhost:3000/create \
Body '{"nombre":"Luna","clase":"Maga","HP":80}'
```

- Respuesta esperada:

```
{
  "type": "success",
  "message": "Ficha creada correctamente"
}
```

9. Despliegue

Volver

- La API se despliega en **Vercel**.
- Se usa el archivo `vercel.json` para definir las rutas y la carpeta raíz.
- Variables de entorno configuradas desde el panel de Vercel.
- Ejemplo de endpoint:

```
https://miapi.vercel.app/api/readall
```

10. Futuras mejoras

Volver

- Implementar autenticación JWT para proteger endpoints.
- Añadir validación de datos con `express-validator`.
- Centralizar manejo de errores.
- Incluir tests automatizados (Jest o Mocha).
- Generar documentación interactiva con Swagger.