Estructura del Backend — Proyecto SnapNation

Índice

Estructura del Backend — Proyecto SnapNation..... 1
1. Visión general.....2
2. Arquitectura general.....2
3. API REST (Capa de Controladores)..... 2
4. Capa de Servicios (Lógica de negocio).....3
5. Capa de Acceso a Datos.....3
6. Base de datos..... 3
7. Seguridad y autenticación.....4
8. Servicios externos.....4
9. Comunicación Frontend–Backend..... 4
10. Conclusión.....4

Javier Manzano Oliveros 2ºDAW

1. Visión general

El backend del proyecto **SnapNation** estará diseñado como una **API REST**, encargada de gestionar la lógica de negocio, el acceso a datos y la seguridad de la aplicación.
El frontend se comunicará con el backend exclusivamente mediante peticiones HTTP, intercambiando datos en formato JSON.

La arquitectura del backend está pensada para ser **modular, escalable y mantenible**, facilitando la ampliación futura del sistema.

---

2. Arquitectura general

El backend se organiza siguiendo una **arquitectura por capas**, separando claramente responsabilidades:

1. **Capa de Controladores (API REST)**
2. **Capa de Servicios (lógica de negocio)**
3. **Capa de Acceso a Datos**
4. **Base de Datos**
   Servicios externos

Esta separación permite mantener el código limpio y desacoplado.

---

3. API REST (Capa de Controladores)

La API REST será el punto de entrada al backend.
Cada endpoint se encarga de recibir peticiones del frontend, validar datos básicos y delegar la lógica a la capa de servicios.

Ejemplos de funcionalidades expuestas por la API:

- Autenticación de usuarios
- Subida de fotografías
- Votación de fotografías
- Obtención de ganadores semanales
- Gestión del perfil de usuario

Los endpoints estarán versionados (`/api/v1`) y devolverán respuestas en formato JSON con códigos HTTP adecuados.

---

4. Capa de Servicios (Lógica de negocio)

La capa de servicios contiene la **lógica principal del sistema**.
Aquí se implementan las reglas de negocio, como:

- Validar si un usuario puede votar una foto
- Comprobar si una foto pertenece al tema semanal
- Controlar límites de participación
- Calcular ganadores semanales
- Aplicar permisos según el rol del usuario

Cada servicio se encarga de una funcionalidad concreta, lo que facilita la reutilización y el mantenimiento del código.

---

5. Capa de Acceso a Datos

Esta capa se encarga de la comunicación directa con la base de datos.

Sus responsabilidades incluyen:

- Consultas de lectura y escritura
- Inserción y actualización de datos
- Gestión de relaciones entre entidades

La capa de servicios **no accede directamente a la base de datos**, sino que utiliza esta capa intermedia para mantener el desacoplamiento.

---

6. Base de datos

La base de datos almacena la información principal del sistema, como:

- Usuarios
- Fotografías
- Votos
- Temas semanales
- Resultados y ganadores

El modelo de datos se diseñará para soportar relaciones entre usuarios y fotografías, así como el sistema de votaciones por comunidad.

---

7. Seguridad y autenticación

El backend implementa un sistema de **autenticación mediante tokens JWT**.

- El usuario se autentica con email y contraseña
- El backend genera un token JWT
- El frontend envía el token en cada petición protegida
- El backend valida el token antes de procesar la solicitud

Esto permite proteger los endpoints y garantizar que solo los usuarios autorizados accedan a determinadas funcionalidades.

---

8. Servicios externos

El backend se comunica con servicios externos cuando es necesario.
En el caso de SnapNation, se utiliza un servicio externo para el **almacenamiento de imágenes**, evitando cargar la base de datos con archivos pesados.

---

9. Comunicación Frontend–Backend

La comunicación entre frontend y backend se realiza mediante:

- Peticiones HTTP (GET, POST, PUT, DELETE)
- Intercambio de datos en formato JSON
- Respuestas con códigos de estado HTTP

Esta comunicación está reflejada en los diagramas de secuencia y componentes del proyecto.

---

10. Conclusión

La estructura del backend de SnapNation está diseñada para ser clara, modular y escalable, separando correctamente las responsabilidades y facilitando el desarrollo y mantenimiento del sistema.
Esta arquitectura permite crecer el proyecto en funcionalidades sin comprometer la estabilidad del sistema.