DOCUMENTACIÓN ESTRUCTURADA DE ABSTRACCIONES

A. Endpoints de Autenticación

Login

Endpoint: POST /auth/login

Método HTTP: POST

Formato de serialización: JSON

Cabeceras de entrada:

- Content-Type: application/json

Cabeceras de salida:

- Content-Type: application/json

Estructura de datos de entrada:

json
```
{
  "email": "string (formato email, máx. 100 caracteres)",
  "password": "string (mínimo 1 carácter)"
}
```

Estructura de datos de salida (caso satisfactorio):

json
```
{
  "message": "string",
  "token": "string",
  "user": {
    "id": "number",
    "name": "string",
    "email": "string",
    "role": "string",
    "creation_date": "string"
  }
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string",
  "errors": {
    "email": "string",
    "password": "string",
    "general": "string"
  }
}
```

Códigos de error posibles: 400, 401, 500

Register

Endpoint: POST /auth/register

Método HTTP: POST

Formato de serialización: JSON

Cabeceras de entrada:

- Content-Type: application/json

Cabeceras de salida:

- Content-Type: application/json

Estructura de datos de entrada:

```
json
{
  "username": "string (1-50 caracteres)",
  "email": "string (formato email, máx. 100)",
  "password": "string (mínimo 8 caracteres, requiere formato)"
}
```

Nota: El campo name se mapea desde username en el backend.

Requisitos de contraseña:

- Mínimo 8 caracteres
- Al menos una letra mayúscula
- Al menos una letra minúscula
- Al menos un número

- Al menos un carácter especial (!@#$%^&*(),.?":{}|<>)

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "message": "string"
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string",
  "errors": {
    "name": "string",
    "email": "string",
    "password": ["string"],
    "general": "string"
  }
}
```

Códigos de error posibles: 400, 500

B. Endpoints de Moldes

Autenticación requerida: Todos los endpoints de moldes requieren un token JWT en el header Authorization: Bearer <token>

GetAllMolds

Endpoint: GET /mold/

Método HTTP: GET

Cabeceras de entrada:

- Content-Type: application/json
- Authorization: Bearer <token>

Cabeceras de salida:

- Content-Type: application/json

Estructura de datos de entrada: No requiere parámetros.

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "molds": [
    {
      "id": "number",
      "name": "string",
      "type": "string",
      "width": "number",
      "height": "number",
      "svg_path": "string",
      "creation_date": "string"
    }
  ]
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string"
}
```

Códigos de error posibles: 401, 500

GetMoldById

Endpoint: GET /mold/:id

Método HTTP: GET

Cabeceras de entrada:

- Content-Type: application/json
- Authorization: Bearer <token>

Estructura de datos de entrada:

- **Parámetro de ruta:** `id` (string de solo dígitos)

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "id": "number",
  "name": "string",
  "type": "string",
  "width": "number",
```

```
  "height": "number",
  "svg_path": "string",
  "creation_date": "string"
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string"
}
```

Códigos de error posibles: 401, 404, 500

CreateMold

Endpoint: POST /mold/

Método HTTP: POST

Cabeceras de entrada:

- Content-Type: application/json
- Authorization: Bearer <token>

Estructura de datos de entrada:

```
json
{
  "name": "string (1-100 caracteres)",
  "type": "string (1-100 caracteres)",
  "width": "number (≥0, opcional)",
  "height": "number (≥0, opcional)",
  "svg_content": "string (contenido del archivo SVG)"
}
```

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "message": "string",
  "mold": {
    "id": "number",
    "name": "string",
    "type": "string",
    "width": "number",
```

```
    "height": "number",
    "svg_path": "string",
    "creation_date": "string"
  }
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string",
  "errors": {}
}
```

Códigos de error posibles: 400, 401, 500

UpdateMold

Endpoint: PUT /mold/:id

Método HTTP: PUT

Cabeceras de entrada:

- Content-Type: application/json
- Authorization: Bearer <token>

Estructura de datos de entrada:

```
json
{
  "name": "string (1-100 caracteres, opcional)",
  "type": "string (1-100 caracteres, opcional)",
  "width": "number (≥0, opcional)",
  "height": "number (≥0, opcional)",
  "svg_content": "string (opcional)"
}
```

Nota: El `id` se envía como parámetro de ruta, no en el body.

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "message": "string",
  "mold": {
    "id": "number",
```

```
    "name": "string",
    "type": "string",
    "width": "number",
    "height": "number",
    "svg_path": "string",
    "creation_date": "string"
  }
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string"
}
```

Códigos de error posibles: 400, 401, 404, 500

DeleteMold

Endpoint: DELETE /mold/:id

Método HTTP: DELETE

Cabeceras de entrada:

- Authorization: Bearer <token>

Estructura de datos de entrada:

- **Parámetro de ruta:** id (string de solo dígitos)

Estructura de datos de salida (caso satisfactorio):

```
json
{
  "message": "string"
}
```

Estructura de datos de salida (caso no satisfactorio):

```
json
{
  "message": "string"
}
```

Códigos de error posibles: 401, 404, 500

C. Sistema WebSocket Colaborativo

Conexión WebSocket

Endpoint: ws://host:port/ws

Protocolo: WebSocket

Autenticación: Token JWT como parámetro

Eventos del Servidor → Cliente

connected

json
```
{
  "type": "connected",
  "clientId": "string",
  "message": "string"
}
```

user_joined

json
```
{
  "type": "user_joined",
  "userId": "number",
  "email": "string"
}
```

user_left

json
```
{
  "type": "user_left",
  "userId": "number",
  "email": "string"
}
```

mold_created

json
```
{
  "type": "mold_created",
  "mold": {
    "id": "number",
    "name": "string",
    "type": "string",
```

```
    "width": "number",
    "height": "number",
    "svg_path": "string",
    "creation_date": "string"
  },
  "userId": "number",
  "email": "string"
}
```

mold_updated

json

```
{
  "type": "mold_updated",
  "mold": {
    "id": "number",
    "name": "string",
    "type": "string",
    "width": "number",
    "height": "number",
    "svg_path": "string",
    "creation_date": "string"
  },
  "userId": "number",
  "email": "string"
}
```

mold_deleted

json

```
{
  "type": "mold_deleted",
  "moldId": "number",
  "userId": "number",
  "email": "string"
}
```

canvas_object_added

json

```
{
  "type": "canvas_object_added",
  "object": {
    "id": "string",
    "moldId": "number",
    "name": "string",
    "x": "number",
    "y": "number",
    "width": "number",
    "height": "number",
    "scale": "number",
```

```
    "rotation": "number",
    "svgPath": "string"
  },
  "userId": "number",
  "email": "string"
}
```

canvas_object_moved

```
json
{
  "type": "canvas_object_moved",
  "objectId": "string",
  "x": "number",
  "y": "number",
  "rotation": "number",
  "scale": "number",
  "userId": "number"
}
```

canvas_object_removed

```
json
{
  "type": "canvas_object_removed",
  "objectId": "string",
  "userId": "number",
  "email": "string"
}
```

canvas_cleared

```
json
{
  "type": "canvas_cleared",
  "userId": "number",
  "email": "string"
}
```

Eventos del Cliente → Servidor

Los clientes pueden enviar los siguientes mensajes para notificar cambios:

- mold_created
- mold_updated
- mold_deleted
- canvas_object_added
- canvas_object_moved
- canvas_object_removed
- canvas_cleared
- ping (para mantener la conexión viva)

D. Archivos Estáticos

Acceso a SVGs

Ruta: /storage/svgs/archivo.svg

Método: GET

Descripción: Los archivos SVG se almacenan en el servidor y se acceden mediante rutas relativas. El campo svg_path de los moldes contiene la ruta relativa desde la raíz del proyecto.

E. Notas Importantes

1. **Rutas del Frontend**: El servidor sirve el mismo archivo index.html para todas las rutas del frontend (/, /auth/login, /auth/register, /dashboard), permitiendo que el router del frontend maneje la navegación.
2. **CORS**: El servidor está configurado para aceptar peticiones desde cualquier origen en desarrollo.
3. **Almacenamiento de SVG**: Los archivos SVG se guardan físicamente en el servidor en storage/svgs/ con nombres únicos generados automáticamente.
4. **Límite de tamaño**: Las peticiones JSON tienen un límite de 10MB configurado en el servidor.