UNIVERSIDAD TECNOLOGICA NACIONAL

DESARROLLO DE SOFTWARE

Documentación Trabajo Práctico Final

2024

Grupo 5

Profesores:

Fernández, José
Villaverde, Jorge

Integrantes:

Arduña Zago, Agustín Juan Luis
Faccini, Lautaro Román
Fernández, Facundo Nahuel
Kinweiler, Víctor
Moselli, Yamil Apas
Schefer, Mauricio Nicolás
Velazco Gez Schegtel, Juan Ignacio

1. INTRODUCCIÓN

El presente documento describe el propósito del proyecto, las funcionalidades que proporcionará al usuario y su principal objetivo, así como las tecnologías y metodologías de desarrollo utilizadas para llevarlo a cabo.

El proyecto abarca un software de gestión y procesamiento de información para el concurso de esculturas “Bienal Internacional de Esculturas”, que se realiza desde 1988 en la ciudad de Resistencia, Argentina. Su objetivo principal es brindar a los usuarios, tanto internos como externos a la organización una interfaz amigable para visualizar toda la información relacionada al certamen.

La organización de la Bienal Internacional de Escultura del Chaco, precisaba de un sistema de gestión que soporte el registro de los eventos, escultores y obras; Como así también un sitio web para que el público general pueda votar durante el evento.

El público objetivo del proyecto es el público general y los administradores del evento.
El público contará con un sistema de autenticación que les permitirá votar sobre sus obras favoritas, en adición a poder visualizar información sobre los eventos, obras y escultores.
Los administradores del evento contarán con funcionalidades tanto para la gestión de eventos como de obras y escultores.

---

2. REQUISITOS

1.1. Requisitos Funcionales

1. Gestión de Eventos: Se desea poder generar eventos futuros (y también poder cargar los eventos pasados para tener el historial) Posibilidad de agregar, ver, modificar y eliminar información sobre cada evento. Detalle del Evento: Información sobre la fecha, lugar, descripción y temática del evento.
2. Gestión de Escultores: Se desea poder mantener la información de los escultores Posibilidad de agregar, ver, modificar y eliminar información de los escultores. Perfil del escultor: Información detallada del escultor incluyendo nombre, biografía, contacto y obras previas.
3. Gestión de Esculturas: Posibilidad de agregar, ver, modificar y eliminar información sobre cada escultura. Temática de la Escultura: Descripción de la temática de cada escultura, fecha de creación, etc.
4. Gestión de Imágenes: Subir y Visualizar Imágenes: Posibilidad de subir y ver fotos de las esculturas en diferentes etapas (antes, durante y después del evento).
5. Aplicación web: pública para visualizar el próximo evento, y los eventos anteriores. Sitio web público para ver los escultores y sus esculturas.
6. Sistema de Votación: Votación por Visitantes: Funcionalidad para que los visitantes puedan votar por sus esculturas favoritas. Deberá estar en el sitio web público. El sistema de votación es con valores del 1 al 5 (el 5 el de mayor puntaje).
7. Autenticación de Votantes: Sistema para asegurar que cada visitante puede votar solo una vez (por ejemplo, a través de una cuenta de usuario o validación por email).
8. Sistema de votación por sitio web público: Haciendo uso de un botón de votar en cada escultor
9. Sistema de votación por QR: En cada escultor estará disponible una tablet/pantalla que visualizará un QR “único” que deben cambiar cada un minuto (para prevenir el uso del QR por fuera del predio, cada minuto cambiará el QR y los anteriores ya no deberán funcionar).

10. PWA (Aplicación Web Progresiva)

1.2. Requisitos No Funcionales

1. Interfaz de Usuario (UI) Adaptable a dispositivos: Se requiere utilizar la aplicación tanto de gestión como de los usuarios finales en diferentes dispositivos (tablet, desktop, móviles)
2. Multiplataforma: Compatibilidad con diferentes navegadores y dispositivos (PC, tablet, móvil).
3. Autenticación y Autorización: Uso de mecanismos seguros para autenticación y autorización de usuarios. Tanto para el área de gestión como para el área de usuarios para la votación.
4. Protección de Datos: Asegurar que los datos de los escultores y visitantes estén protegidos contra accesos no autorizados.
5. Tiempo de Respuesta: Garantizar tiempos de respuesta rápidos en la carga de vistas/páginas y procesamiento de datos.
6. Optimización de Imágenes: Asegurar que las fotos subidas estén optimizadas para una carga rápida sin pérdida de calidad significativa.
7. Usabilidad Interfaz Intuitiva: Diseño de una interfaz que sea fácil de usar tanto para administradores como para visitantes.
8. Accesibilidad: Asegurar que la aplicación sea accesible para usuarios con discapacidades.
9. Integración con Redes Sociales: Posibilidad de compartir eventos y esculturas en redes sociales.
10. Sistema de validación de votantes: para evitar fraudes, posiblemente mediante la integración de un sistema de autenticación externo o captcha.

1.3. Requisitos del Sistema

Para el entorno de desarrollo y producción:

Procesador: CPU de 2 núcleos o más
Memoria RAM: 8 GB (mínimo recomendado).
Almacenamiento:20 GB de espacio libre en disco para instalar dependencias
Red: Conexión a internet estable para descargar dependencias, usar API o trabajar con Google Cloud Storage.
Sistema operativo:

- Windows 10 o superior (64 bits).
- macOS Catalina o superior.
- Linux (Ubuntu 20.04 o superior).

Navegador web moderno: Chrome, Firefox, Edge o Safari actualizado para la vista previa de React.
Acceso a Google Cloud Storage (requiere configuración previa y conexión a internet).

Para el acceso de usuarios comunes:

Computadora de escritorio o portátil y dispositivos móviles:

- Procesador: CPU con 2 núcleos
- Memoria RAM: 2 GB (mínimo), 4 GB recomendado para una experiencia fluida.

Navegador compatible:

- Google Chrome (versión 90 o superior).
- Mozilla Firefox (versión 90 o superior).
- Microsoft Edge (versión 90 o superior).
- Safari (versión 13 o superior para iOS/macOS).

Sistema operativo:

- Windows 7/8/10/11, macOS (10.13 High Sierra o superior), Linux (distribuciones modernas).
- Android (7.0 Nougat o superior) o iOS (13.0 o superior).

---

3. GUIA DE INSTALACION

Dependencias utilizadas y requeridas para el correcto funcionamiento y despliegue de la aplicación:

Git/Github (versión 2.47.1): Sistema de control de versiones que utilizamos para llevar un registro mucho más ordenado de los cambios del proyecto y así permitir el desarrollo conjunto desde distintos ordenadores.
Node.js (versión 20.17.0): Entorno de desarrollo utilizado para desarrollar la aplicación
MySQL (versión 8.0.39): Para el manejo de la base de datos.
Google Cloud Storage (versión 7.14.0): Para almacenar las imágenes provistas por la base de datos.
Express (versión 4.21.1): Framework para utilizar en conjunto con Node.js
React (versión 18.3.1): Biblioteca de javascript para generar interfaces de usuario de forma sencilla
Vite (versión 6.0.2): Para utilizar en conjunto con React y agilizar el proceso de desarrollo y despliegue.
Tailwind CSS (4.0.0): Framework para manipular estilos CSS de forma más sencilla y rápida.

Pasos de Instalación: Instrucciones paso a paso para instalar y configurar el software.

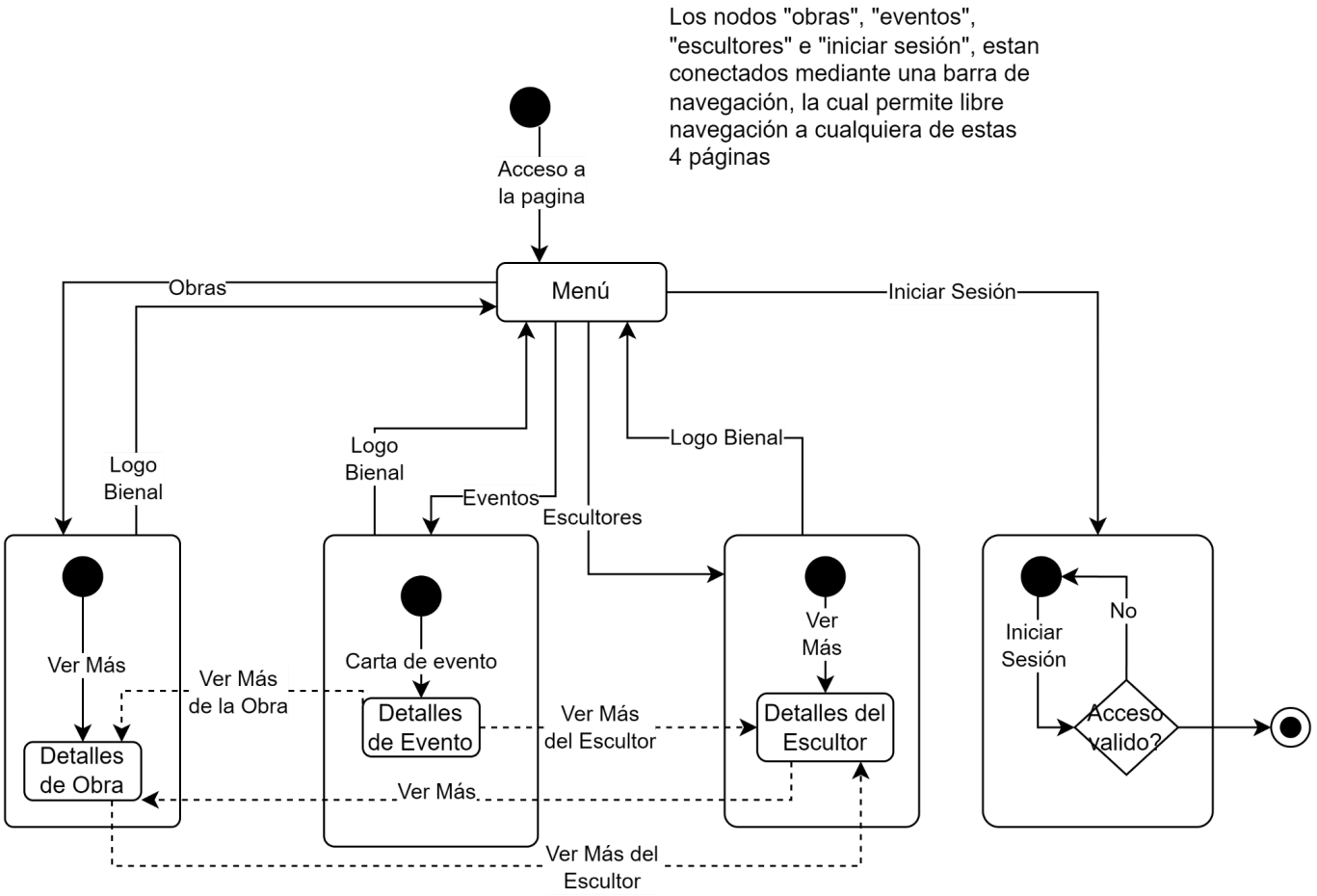
1. Ejecutar el comando "*git clone https://github.com/FRRe-DS/2024-05-TPI.git*"
2. En las carpetas raíz y cliente ejecutar el comando "*npm install*" para instalar las dependencias necesarias.
3. Crear un .env en /client con la siguiente información:
   VITE_HOST: Tu direccion IP (opcional)
   VITE_SERVER_PORT: El puerto donde se está ejecutando el servidor (opcional)
   VITE_PORT: El puerto donde se ejecutará el cliente (opcional)
4. Crear un archivo "*.env*" en la carpeta raíz y pegar los siguientes datos:
   DB_HOST: Host de la base de datos
   DB_PORT: Puerto de la base de datos
   DB_USER: Usuario de la base de datos
   DB_PASSWORD: Contraseña del usuario de la base de datos
   DB_DATABASE: Nombre de la base de datos
   HOST: Tu direccion IP (opcional)
   PORT: El puerto donde se ejecutará el cliente (opcional)
   TOKEN_SECRET: Clave para los token de autenticación
   SECRET_KEY: Clave para los token de los QR
5. Ejecutar "*npm run dev*" en las carpetas *server* y *client*

4. USO DEL SOFTWARE

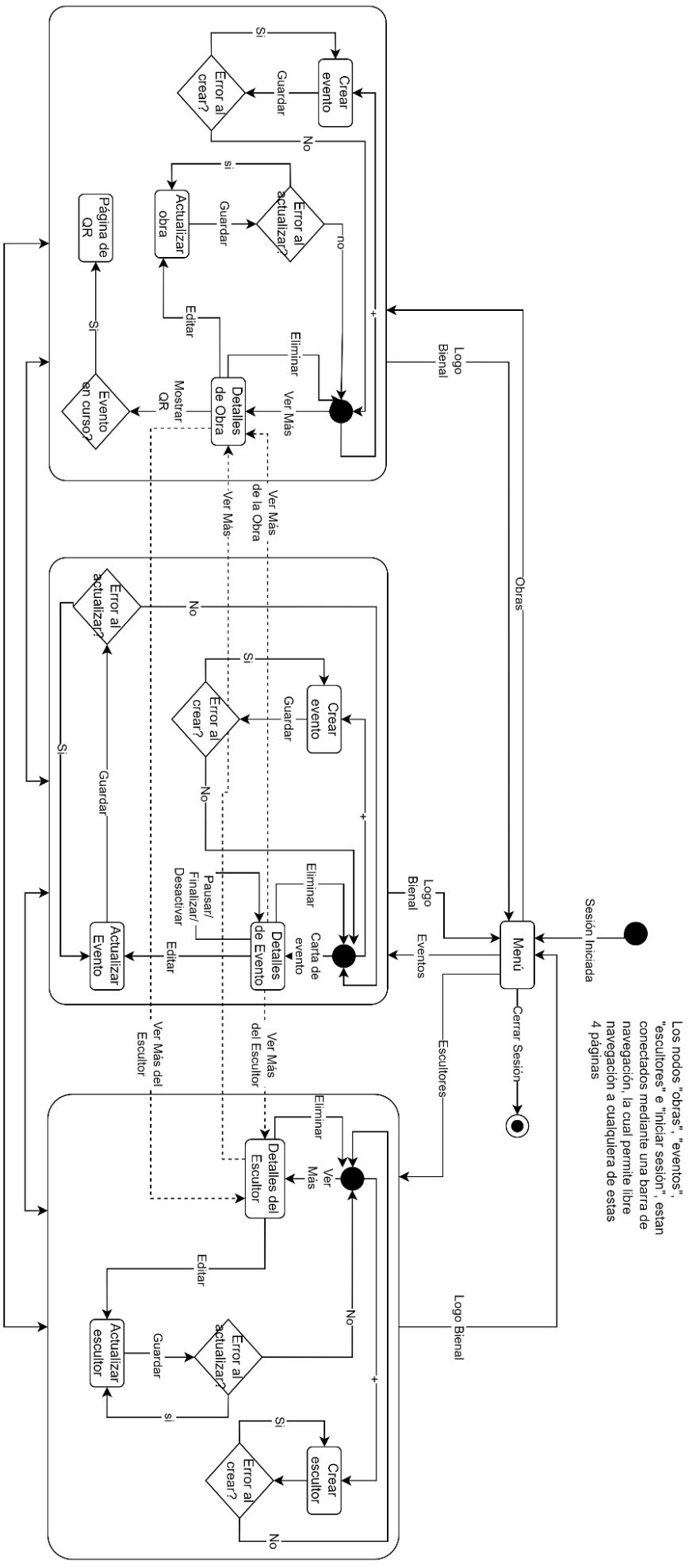
Guía de Usuario

Diagrama de navegación

1. Usuario:



2. Admin:


Los nodos "obras", "eventos", "escultores" e "iniciar sesión", estan conectados mediante una barra de navegación, la cual permite libre navegación a cualquiera de estas 4 páginas
Sesión Iniciada
Menú
Cerrar Sesión
Obras
Eventos
Escultores
Logo Bienal
Crear evento
Error al crear?
Si
No
Guardar
Actualizar obra
Error al actualizar?
si
no
Página de QR
Editar
Eliminar
Ver Más
Detalles de Obra
Evento en curso?
Mostrar QR
Ver Más de la Obra
Pausar/ Finalizar/ Desactivar
Carta de evento
Detalles de Evento
Actualizar Evento
Ver Más del Escultor
Detalles del Escultor
Actualizar escultor
Crear escultor

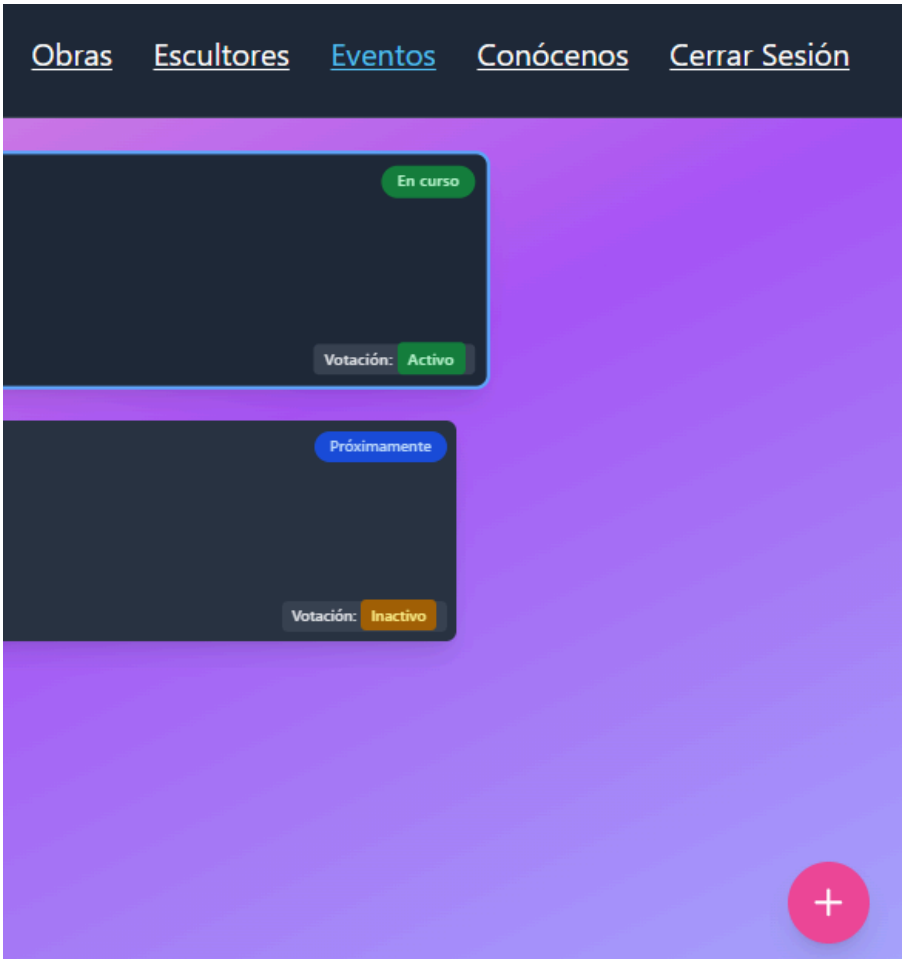
Casos de Uso Comunes

Crear un evento:

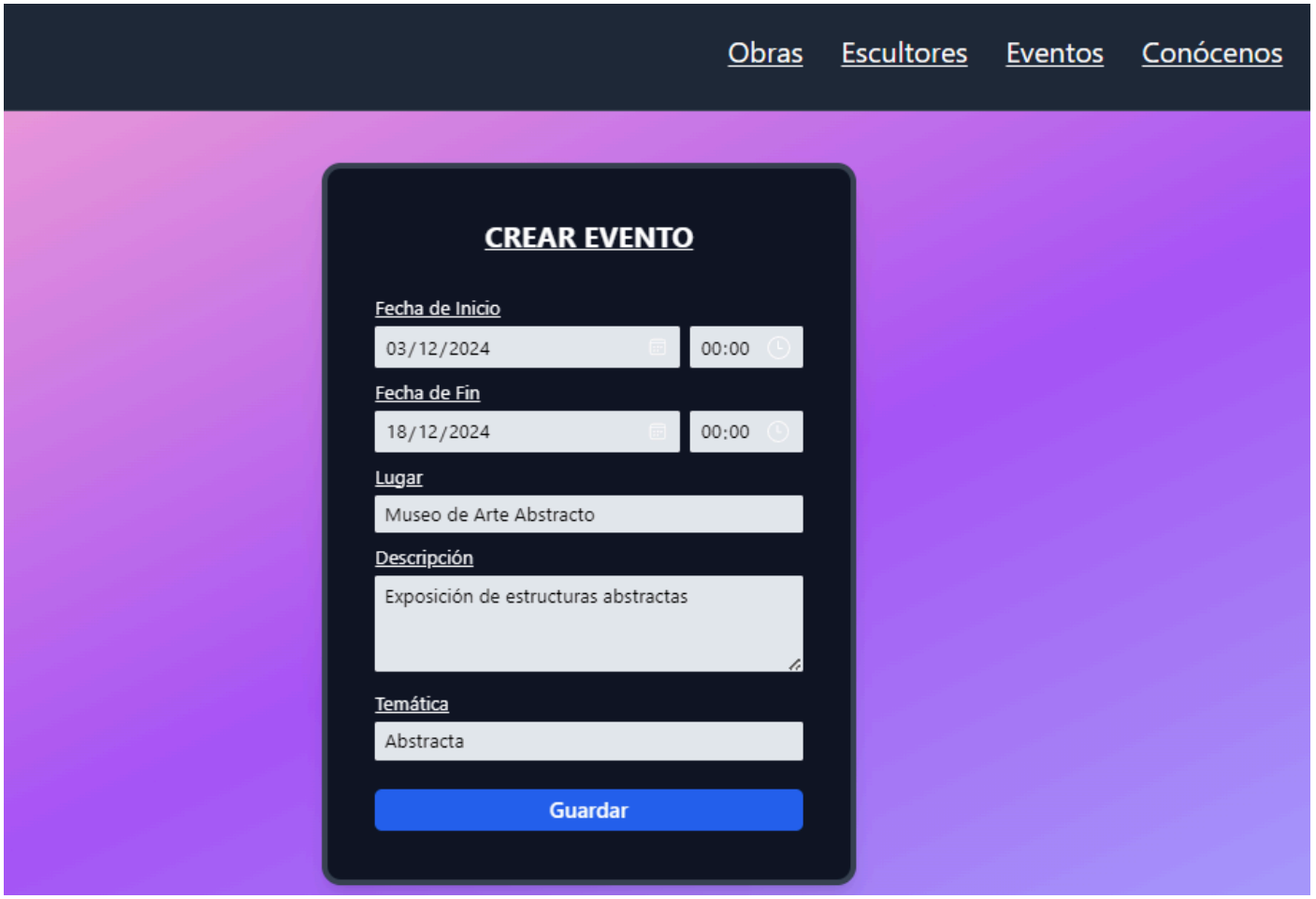
Pre-requisito: Autenticado con privilegios de administrador

1. Seleccionamos la opción “Eventos”

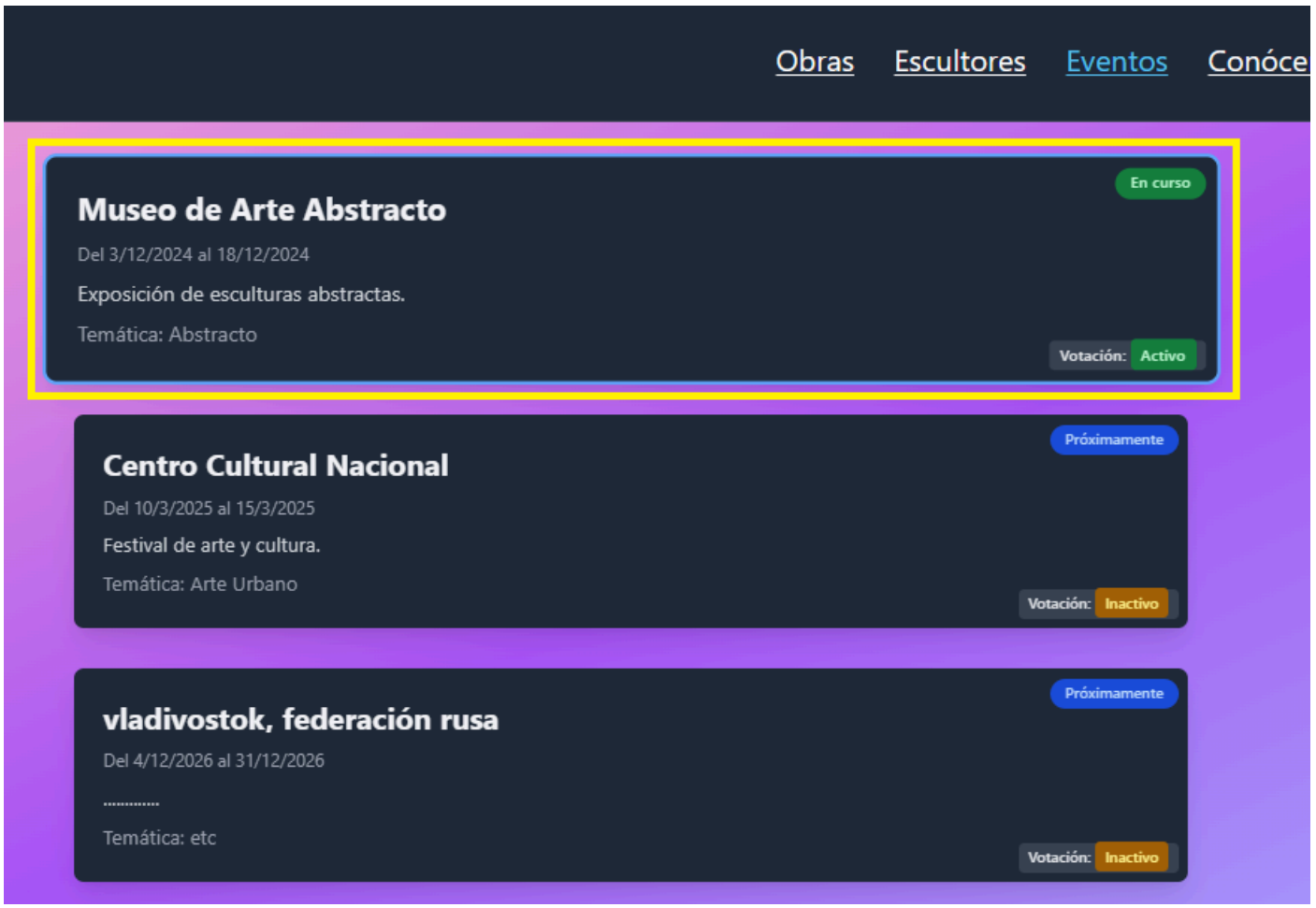
2. Seleccionamos el signo “+” en la esquina inferior derecha de la pantalla



3. Rellenamos los campos y seleccionamos la opción “guardar”



4. El evento aparecerá como “en curso”, “proximamente” o “finalizado” según los parámetros establecidos.

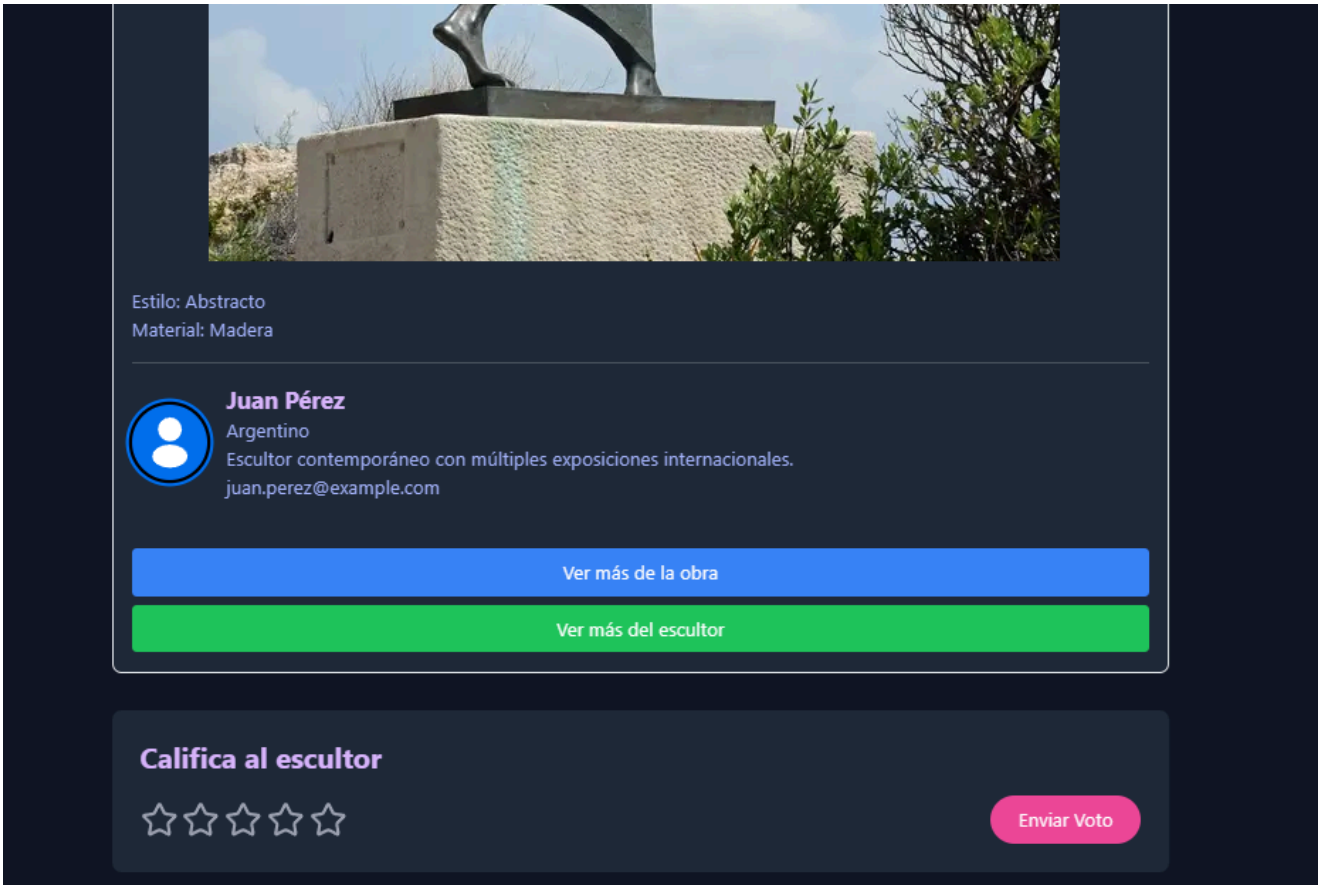


Votación:

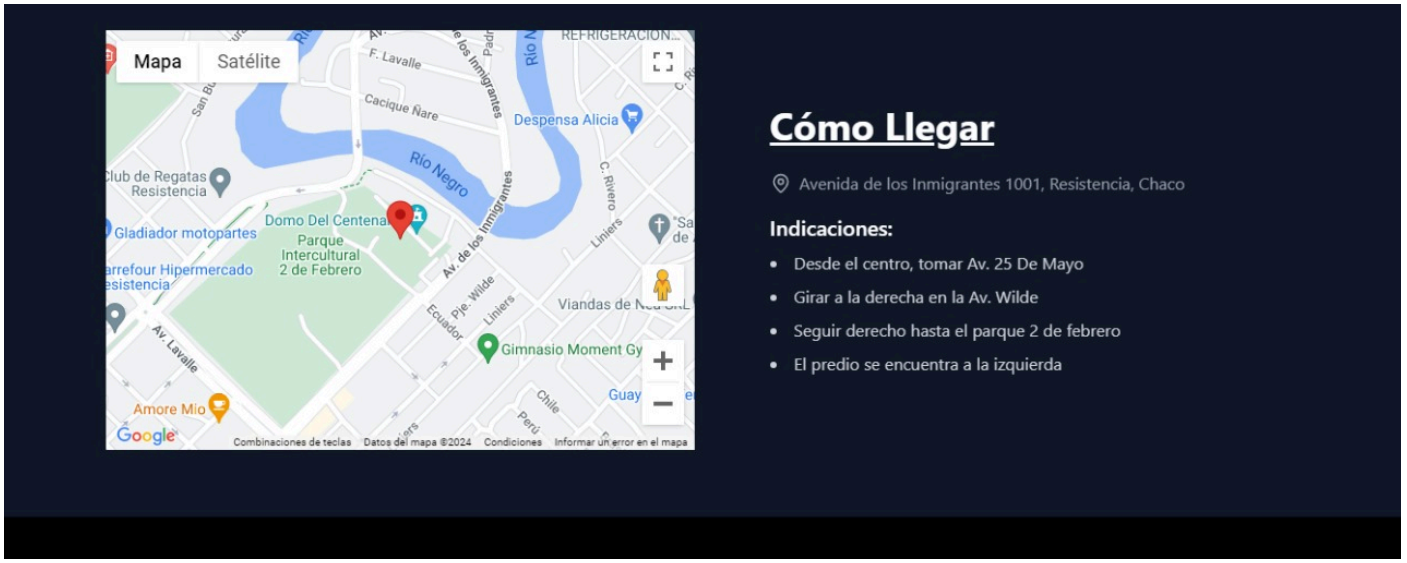
Pre-requisito: Usuario autenticado en el sistema y un token QR proporcionado por los administradores.

1. El usuario escanea el token QR

2. El usuario accede a la página de votación, donde se despliega información como la obra a la que va a votar y el escultor que la fabricó.



3. El usuario califica a la obra y selecciona la opción “Enviar Voto”
    3.1. Si el token QR se renueva antes de que el usuario haya enviado su voto, el sistema dará error.
    3.2. Si el usuario ya ha votado por la obra en cuestion anteriormente, el sistema dará error.



---

5. ARQUITECTURA Y DISEÑO

Diagrama de Arquitectura

Este proyecto utiliza una arquitectura cliente-servidor basada en Node.js para el backend, y una base de datos SQL. Sigue un enfoque monolítico modular, donde las diferentes responsabilidades del backend están organizadas en módulos para facilitar su mantenimiento y escalabilidad.

El backend expone la API para la interacción con el cliente y otros servicios, siguiendo el estilo de servicios REST. La estructura también respeta la separación de responsabilidades, donde los controladores manejan la lógica de negocio e interactúan con la base de datos.

Estructura del Proyecto

Las carpetas se distribuyen de la siguiente manera:

Carpetas principales:

- src/: Contiene el código fuente principal del proyecto.
    - routes/: Define las rutas para el servidor, indicando las diferentes entradas al sistema.
    - controllers/: Encargados de la lógica de negocio y la conexión entre rutas y modelos.
    - services/: Implementa lógica adicional o abstracciones sobre funcionalidades específicas, como la lógica de QR.
    - middlewares/: Principalmente esquemas de validación de autenticaciones (login).
- database/: Contiene scripts SQL para la creación y configuración de la base de datos.
- public/ o static/ (si existe): Carpeta destinada a archivos estáticos que el cliente puede requerir directamente.

Archivos principales:

- package.json: Define las dependencias y scripts del proyecto.

Decisiones de Diseño

División modular en carpetas:

Se optó por organizar el proyecto en carpetas que representan distintas responsabilidades, como rutas, controladores y modelos. Esta estructura sigue los principios de separación de responsabilidades, lo que facilita la escalabilidad del sistema. Además, permite que varios desarrolladores trabajen simultáneamente en diferentes áreas del código sin generar conflictos.

Uso de una base de datos SQL:

Se eligió una base de datos relacional debido a su enfoque estructurado, que resulta ideal para gestionar datos bien definidos. Esto garantiza la integridad de los datos y permite realizar consultas complejas de manera eficiente, adaptándose a las necesidades del proyecto.

QR proporcionado por los administradores:

Se optó por permitir que solo los administradores tengan acceso al QR, el cual se actualizará cada cierto tiempo, estos deberán ser configurados por los mismos administradores para que los usuarios puedan escanearlo en puntos comunes (totems). De esta forma se motiva al usuario a votar in-situ y se minimiza cualquier tipo de fraude (mismo usuario votando desde diferentes dispositivos).

Dependencias

Node.js: Entorno de ejecución para el backend.

Express: Framework para manejar las rutas y middleware.

MySQL: Librería para interactuar con la base de datos relacional.

6. API

Referencia de la API

La API cuenta con múltiples endpoints organizados por áreas funcionales: autenticación, escultores, eventos, obras y votos.

Autenticación

`POST /register`
Registra un nuevo usuario.

- Parámetros:
  - `user`: Objeto JSON con información del usuario.
- Respuesta:
  - `200 OK`: Usuario registrado exitosamente.
  - `400 Bad Request`: Datos de registro inválidos.

`POST /login`
Inicia sesión de un usuario.

- Parámetros:
  - `user`: Objeto JSON con credenciales del usuario.
- Respuesta:
  - `200 OK`: Usuario autenticado.
  - `401 Unauthorized`: Credenciales incorrectas.

`GET /verify`
Verifica si el token de autenticación es válido.

- Respuesta:
  - `200 OK`: Token válido.
  - `401 Unauthorized`: Token inválido o expirado.

Escultores

`GET /escultores`
Obtiene la lista de escultores.

- Respuesta:
  - `200 OK`: Lista de escultores.

`POST /escultores`
Crea un nuevo escultor.

- Parámetros:
  - `escultor`: Objeto JSON con los datos del escultor.
  - `foto_perfil`: Imagen opcional del escultor (multipart/form-data).
- Respuesta:
  - `201 Created`: Escultor creado.

`DELETE /escultores/:id`
Elimina un escultor por ID.

- Respuesta:
  - `204 No Content`: Escultor eliminado.

Eventos

`GET /eventos`
Lista todos los eventos.

- Respuesta:
  - `200 OK`: Lista de eventos.

`POST /eventos`
Crea un nuevo evento.

- Parámetros:
  - `evento`: Objeto JSON con los datos del evento.
- Respuesta:
  - `201 Created`: Evento creado.

`POST /eventos/activar/:id`
Activa un evento por ID.

- Respuesta:
  - `200 OK`: Evento activado.

Obras

`GET /obras`
Lista todas las obras.

- Respuesta:
  - `200 OK`: Lista de obras.

POST /obras
Crea una nueva obra.

- Parámetros:
  - obra: Objeto JSON con los datos de la obra.
  - images: Array de imágenes asociadas (multipart/form-data).
- Respuesta:
  - 201 Created: Obra creada.

GET /obras/:id
Obtiene los detalles de una obra por ID.

- Respuesta:
  - 200 OK: Detalles de la obra.

Votaciones

POST /vota
Registra un nuevo voto.

- Parámetros:
  - voto: Objeto JSON con los datos del voto.
- Respuesta:
  - 201 Created: Voto registrado.

---

7. MANTENIMIENTO

Actualizaciones:

La única forma de actualizar el software es mediante comandos de git (git pull)

Resolución de Problemas:

Mantener actualizada la base de datos para que esta no interfiera con el funcionamiento del software.

Soporte:

Issues · FRRe-DS/2024-05-TPI

---

8. SEGURIDAD

La arquitectura cliente-servidor, como ya se ha descrito anteriormente facilita la separación de responsabilidades entre el frontend, el backend, y la base de datos. Esta arquitectura presenta varias ventajas en términos de seguridad, como la separación de responsabilidades utilizando el estilo MVC.

Frontend: Solo interactúa con el backend, limitando la exposición directa a datos sensibles. Las operaciones críticas se realizan en el servidor.

Backend: Maneja la lógica de negocio y la comunicación con la base de datos, restringiendo el acceso directo a los datos por parte del cliente. El backend actúa como intermediario, lo que permite implementar autenticación y autorización centralizadas. Esto ayuda a controlar quién puede acceder a qué recursos, utilizando mecanismos como JWT, OAuth o API Keys.

Base de Datos: Está aislada del acceso directo del cliente, lo que reduce la probabilidad de ataques como inyecciones SQL si el backend está bien configurado.

También garantiza una capa de seguridad en el backend usando:

Limitación de tasas (Rate Limiting): Previene ataques de fuerza bruta y DoS.

Cabeceras de seguridad (HTTP Security Headers): Mejora la protección contra ataques basados en navegadores.

Autenticación fuerte: Como contraseñas encriptadas, OAuth 2.0 o autenticación multifactor.

La base de datos no está directamente expuesta al cliente. Solo el backend tiene acceso a ella, lo que minimiza la superficie de ataque.

Herramientas como Express.js (backend) y React (frontend) incluyen medidas de seguridad por defecto, como la prevención de vulnerabilidades de XSS y CSRF.

El modelo es compatible con la implementación de HTTPS, lo que asegura la transmisión de datos cifrados entre cliente y servidor. Esto previene ataques de tipo "Man in the Middle" (MITM).

Cualquier vulnerabilidad encontrada debe informarse directamente a los desarrolladores.