### 1. Set Up a Database (for storing metadata)

We’ll need a database to store metadata about your paintings, like title, description, and collection. For simplicity, let's use **MongoDB**, which works great with Node.js and is scalable.

#### Steps to Set Up MongoDB:

* **Create an account** on [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) if you don't already have one.
* Create a **new cluster** in MongoDB Atlas (you can use the free tier).
* **Create a database** for your paintings (e.g., lirinart).
* **Create a collection** within the database (e.g., paintings).
* Get the **connection string** for your MongoDB database.

### 2. Upload Paintings to Cloud Storage

You’ll need to upload your artwork to cloud storage, and I recommend using **Amazon S3** for this.

#### Steps to Set Up Amazon S3:

* **Create an AWS account** if you don’t already have one.
* **Create a new S3 bucket** (e.g., lirinart-bucket).
* **Upload your images** to the bucket.
* **Set permissions** to allow private access (only via signed URLs or via your server).
* You’ll get a unique URL for each image, but to keep things secure, we’ll use **pre-signed URLs**.

### 3. Create the Server API (using Node.js)

We’ll set up a basic Node.js server using **Express** that will handle requests from the frontend, serve metadata from MongoDB, and securely link to the images in S3.

#### Steps to Set Up the Server:

* **Initialize your Node.js app** (if not already done):
* bash
* Copy
* mkdir lirinart-server
* cd lirinart-server
* npm init -y
* npm install express mongoose aws-sdk
* **Create the server** (e.g., server.js):
* js
* Copy
* const express = require('express');
* const mongoose = require('mongoose');
* const AWS = require('aws-sdk');
* const app = express();
* const port = process.env.PORT || 5000;
* // MongoDB setup
* mongoose.connect('YOUR\_MONGO\_DB\_CONNECTION\_STRING', { useNewUrlParser: true, useUnifiedTopology: true });
* const Painting = mongoose.model('Painting', new mongoose.Schema({
* title: String,
* description: String,
* image\_url: String,
* }));
* // AWS S3 setup
* const s3 = new AWS.S3();
* const BUCKET\_NAME = 'your-bucket-name';
* // API route to get paintings
* app.get('/api/paintings', async (req, res) => {
* try {
* const paintings = await Painting.find();
* res.json(paintings);
* } catch (err) {
* res.status(500).json({ error: 'Failed to fetch paintings' });
* }
* });
* // Start server
* app.listen(port, () => console.log(`Server running on port ${port}`));

### 4. Configure Secure Access to Images

To protect your images, you can use **pre-signed URLs** with AWS S3. These URLs grant temporary access to your images.

Here’s an example of how to generate a pre-signed URL for an image in S3:

js

Copy

const generatePresignedUrl = (fileName) => {

const params = {

Bucket: BUCKET\_NAME,

Key: fileName,

Expires: 60 \* 5, // URL expires in 5 minutes

};

return s3.getSignedUrl('getObject', params);

};

// Example usage

const imageUrl = generatePresignedUrl('path/to/your-image.jpg');

console.log(imageUrl);

### 5. Frontend (React or Static Site with Dynamic Content)

You’ll need a frontend that communicates with the API to fetch the paintings and display them. We can use **React** or any static site generator (like **Next.js**, which works seamlessly with Vercel and dynamic content).

Here’s an example for React:

* **Install React**:
* bash
* Copy
* npx create-react-app lirinart-frontend
* cd lirinart-frontend
* npm install axios
* **React Component to Fetch Paintings**:
* js
* Copy
* import React, { useEffect, useState } from 'react';
* import axios from 'axios';
* function Gallery() {
* const [paintings, setPaintings] = useState([]);
* useEffect(() => {
* axios.get('YOUR\_SERVER\_URL/api/paintings')
* .then((response) => {
* setPaintings(response.data);
* })
* .catch((error) => console.error(error));
* }, []);
* return (
* <div>
* <h1>Gallery</h1>
* <div className="gallery">
* {paintings.map((painting) => (
* <div key={painting.\_id} className="painting">
* <h2>{painting.title}</h2>
* <p>{painting.description}</p>
* <img src={painting.image\_url} alt={painting.title} />
* </div>
* ))}
* </div>
* </div>
* );
* }
* export default Gallery;

### 6. Deploying to Vercel

* **Backend (API)**: You can deploy the Node.js server on **Vercel**. Create a vercel.json file and set the builds configuration to recognize your serverless API.
* json
* Copy
* {
* "builds": [
* {
* "src": "server.js",
* "use": "@vercel/node"
* }
* ]
* }
* **Frontend**: Deploy your React (or Next.js) app directly to Vercel. Connect the GitHub repository to Vercel, and it will automatically deploy whenever you push changes to your repo.

### Next Steps:

1. **Complete the backend setup**: Create the database, load the images, and set up secure access.
2. **Set up the frontend**: Display paintings with the data from the backend, including protected image links.
3. **Deploy**: Deploy the backend and frontend to Vercel.
4. **Test**: Make sure everything is working, especially image access and data fetching.
5. **Security**: Ensure that your image links and API endpoints are secure, and only authorized users can access sensitive data.

This is a good starting point to build your serverless architecture. If you need more detailed steps or explanations on any part, feel free to ask!