Here's a **detailed breakdown** of everything we did from **start to finish** to build the **SnapSound** project—an **image-based song recommendation app**.

**SnapSound Project: From Start to Finish**

**📌 Project Overview**

SnapSound is a **React + Node.js** application that analyzes an image to detect mood and recommends songs based on the detected emotion. It integrates **Clarifai API** for image analysis and **Spotify API** for song recommendations.

**📁 Project Setup**

**1️⃣ Created the Project Folder Structure**

We started by creating a dedicated project folder and structuring it properly for both frontend and backend:

mkdir snapSound

cd snapSound

mkdir backend frontend

**🔹 Backend Development (Node.js, Express.js)**

**2️⃣ Initialized the Backend**

Inside the backend/ folder, we initialized a Node.js project and installed required dependencies:

cd backend

npm init -y

npm install express cors dotenv axios multer body-parser

📌 **Tech Stack Used:**

* **Express.js**: Backend framework
* **CORS**: To handle cross-origin requests
* **dotenv**: To manage API keys securely
* **Axios**: For making HTTP requests
* **Multer**: For handling image uploads
* **Body-parser**: To handle JSON requests

**3️⃣ Setting Up server.js and app.js**

We decided to use **two files** for better modularity:

1. server.js → Handles **image analysis & Spotify API**
2. app.js → Manages **server setup and API routing**

**📄 server.js**

* Accepts image in **base64 format**
* Sends the image to **Clarifai API** to detect mood
* Fetches songs from **Spotify API** based on detected mood
* Returns song recommendations to the frontend

**📄 app.js**

* Sets up **Express.js server**
* Handles API routes and ensures the backend runs properly

**4️⃣ Implemented Clarifai API for Image Analysis**

We signed up on **Clarifai**, created a model for **general image recognition**, and got an **API key**.  
Then, we integrated it into server.js using **base64 image format** for better handling.

**Problem Faced:** Initially, we sent an image URL, but it caused **issues with local uploads**.  
**Solution:** Switched to **base64 encoding** and sent the raw image bytes.

**5️⃣ Integrated Spotify API for Song Recommendations**

* We created a **Spotify Developer account**
* Registered a new app to get **Client ID & Secret**
* Implemented **OAuth2 authentication** to get an **access token**
* Used **Spotify's Search API** to find songs related to the detected mood

**6️⃣ Fixed Backend Issues**

* **Error: "PayloadTooLargeError: request entity too large"**  
  **Cause:** Large base64 images exceeded default request size  
  **Fix:** Increased Express.js **body size limit**:
* app.use(express.json({ limit: "10mb" }));
* **Error: "Cannot GET /"**  
  **Cause:** The server lacked a default route  
  **Fix:** Added a welcome message in app.js:
* app.get("/", (req, res) => res.send("Welcome to the SnapSound API!"));

**🔹 Frontend Development (React.js)**

**7️⃣ Initialized the React Frontend**

Inside the frontend/ folder:

cd ../frontend

npx create-react-app .

npm install axios

📌 **Tech Stack Used:**

* **React.js**: Frontend framework
* **Axios**: To communicate with the backend

**8️⃣ Implemented Image Upload and Base64 Conversion**

We created ImageUploader.js where:

* Users **upload an image**
* It's **converted to base64**
* Sent to the backend for **mood detection**
* Songs are displayed based on the **detected mood**

**9️⃣ Connected Frontend & Backend**

* Configured **CORS** in Express.js (app.js)
* Used fetch in React to send the **base64 image** to the backend

**🔹 Problems Faced & Solutions**

| **Problem** | **Cause** | **Solution** |
| --- | --- | --- |
| **Backend not responding** | Server not started | Started backend with node app.js |
| **Frontend not fetching data** | Incorrect API URL | Used "http://localhost:5000/analyze-image" |
| **PayloadTooLargeError** | Image too big | Increased body-parser size |
| **Clarifai API error** | Incorrect request format | Used base64 instead of URL |
| **Spotify API not returning songs** | Expired token | Implemented proper OAuth2 token flow |

**🔹 Final Testing & Deployment**

**🔟 Testing Locally**

We ran both frontend and backend simultaneously:

**Backend:**

cd backend

node app.js

**Frontend:**

cd frontend

npm start

**1️⃣1️⃣ Deployment Plan**

* **Frontend**: Will be deployed on **Vercel/Netlify**
* **Backend**: Can be deployed on **Render/Heroku**
* **Environment Variables**: .env file should be properly handled to secure API keys

**✨ Final Thoughts**

**📌 What We Achieved**

✅ Built a **full-stack web app** with **React.js & Node.js**  
✅ Implemented **Clarifai API** for **image-based mood detection**  
✅ Integrated **Spotify API** for **song recommendations**  
✅ Overcame multiple **backend & frontend challenges**  
✅ Established **a smooth connection** between frontend & backend

This project is now a **solid base** for further improvements like:  
🔹 **More advanced mood analysis**  
🔹 **Better song filtering & recommendations**  
🔹 **Deploying for public use**

**🔹 Full Project Structure**

snapSound/

│── backend/

│ ├── server.js

│ ├── app.js

│ ├── .env

│ ├── package.json

│── frontend/

│ ├── src/

│ │ ├── components/

│ │ │ ├── ImageUploader.js

│ │ ├── App.js

│ ├── package.json

│── README.md

**🚀 Next Steps**

Now that everything is working, you can:  
🔹 Deploy it online  
🔹 Improve the UI/UX  
🔹 Expand features like **playlist recommendations**

Let me know if you need **deployment help** or **further improvements**! 🚀🎶

**🔹 Additional Challenges & Solutions in SnapSound**

**1️⃣ CORS (Cross-Origin Resource Sharing) Issue**

**Problem:**

When our React frontend (running on localhost:3000) tried to fetch data from the backend (localhost:5000), the browser blocked the request due to **CORS restrictions.**

**Error Message:**

Access to fetch at 'http://localhost:5000/analyze-image' from origin 'http://localhost:3000' has been blocked by CORS policy

**Cause:**

By default, browsers block requests between **different origins** for security reasons.

**Solution:**

We used **CORS middleware** in Express (app.js) to allow cross-origin requests:

📄 **app.js (Backend)**

const cors = require("cors");

app.use(cors({

origin: "http://localhost:3000", // Allow frontend requests

methods: ["GET", "POST"],

credentials: true,

}));

🔹 This allowed our frontend to **communicate** with the backend **without CORS issues.**

**2️⃣ Blob Protocol Error & Fixing Image Upload**

**Problem:**

Initially, when we uploaded an image in our **React frontend**, we used URL.createObjectURL(file). However, when sending this **Blob URL** to the backend, it resulted in an **error** because the backend couldn't process it.

**Error Message:**

Failed to load resource: net::ERR\_INVALID\_URL

**Cause:**

* URL.createObjectURL(file) generates a **temporary** URL that only works in the browser.
* When we sent this URL to the backend, **it couldn't fetch the image** because the backend expects **raw image data**, not a browser-generated Blob URL.

**Solution:**

We **converted** the image to **Base64 format** before sending it to the backend.

📄 **ImageUploader.js (Frontend)**

const handleImageUpload = (event) => {

const file = event.target.files[0];

if (file) {

setImage(URL.createObjectURL(file)); // For preview

// Convert to Base64

const reader = new FileReader();

reader.readAsDataURL(file);

reader.onloadend = () => {

const base64String = reader.result.split(",")[1]; // Remove metadata

setBase64Image(base64String);

};

}

};

🔹 This ensured that the backend **received raw image data** instead of a **Blob URL.**

**3️⃣ PayloadTooLargeError (Large Image Handling in Express.js)**

**Problem:**

When sending large base64 images, we got an **error** in the backend:

**Error Message:**

PayloadTooLargeError: request entity too large

**Cause:**

By default, **Express limits request sizes** for security reasons. Our **base64 images exceeded this limit** and caused Express to reject the request.

**Solution:**

We **increased the request size limit** in app.js by setting a higher **body-parser limit**.

📄 **app.js (Backend)**

app.use(express.json({ limit: "10mb" })); // Allow larger payloads

app.use(express.urlencoded({ extended: true, limit: "10mb" }));

🔹 This **allowed larger images** to be sent without errors.

**4️⃣ JSON Formatting Issue in Backend Requests**

**Problem:**

When testing API calls using curl or Postman, sometimes the backend **could not parse JSON correctly** and threw an error.

**Error Message:**

SyntaxError: Unexpected token in JSON

**Cause:**

* The request body was **not formatted properly**.
* We were sending **invalid JSON**, missing Content-Type: application/json header.

**Solution:**

We ensured that:  
✅ The **frontend** always sent properly formatted JSON  
✅ The **backend** correctly parsed the request body

📄 **server.js (Backend)**

app.post("/analyze-image", (req, res) => {

try {

const { imageBytes } = req.body; // Extract Base64 image

if (!imageBytes) {

return res.status(400).json({ error: "No image provided" });

}

// Process the image...

res.json({ mood: "Happy", songs: [] }); // Dummy response for testing

} catch (error) {

console.error("Error:", error);

res.status(500).json({ error: "Server error" });

}

});

📄 **ImageUploader.js (Frontend)**

const analyzeImage = async () => {

if (!base64Image) {

alert("Please upload an image first!");

return;

}

try {

const response = await fetch("http://localhost:5000/analyze-image", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ imageBytes: base64Image }),

});

const data = await response.json();

setMood(data.mood);

setSongs(data.songs);

} catch (error) {

console.error("Error analyzing image:", error);

}

};

🔹 Now, both the frontend and backend **ensure valid JSON formatting**.

**✅ Final Summary of Challenges & Solutions**

| **Issue** | **Cause** | **Solution** |
| --- | --- | --- |
| **CORS error** | Backend blocks requests from frontend | Used cors() middleware in Express |
| **Blob protocol error** | Backend can't read createObjectURL images | Converted image to **Base64** before sending |
| **PayloadTooLargeError** | Base64 images too large for Express | Increased body-parser limit to "10mb" |
| **JSON parsing error** | Invalid JSON format in request body | Ensured **correct JSON structure & headers** |

**🚀 Final Thoughts**

With **all these issues fixed**, our project is now **fully functional and ready for deployment!** 🚀🎵

Let me know if you want help with **deployment or further improvements!** 🔥