🚀 Master Blueprint: KTYM Lead Generation Engine

1. Project Overview

Goal: Build a robust, local-first web application to scrape business leads from Google Maps based on user-defined keywords and locations. **Output:** A Dashboard UI to trigger scrapes, view real-time logs, display results in a table, and export data to CSV. Data is persisted to MongoDB.

2. Technical Stack

- **Framework:** Next.js (Pages Router)
- **Language:** JavaScript (Node.js)
- **Scraping Engine:** Puppeteer (Headless Chrome)
- **Database:** MongoDB (via Mongoose)
- **Styling:** Tailwind CSS + Lucide React (Icons)
- **Environment:** Local Machine (Required for Puppeteer execution)

3. Directory Structure

The AI Agent must ensure the project follows this exact structure:

```
ktym-leads/
├── .env.local              # Environment variables (Sensitive)
├── package.json            # Dependencies
├── tailwind.config.js      # CSS Configuration
├── lib/
│   ├── dbConnect.js        # Singleton Database Connection
│   └── scraper.js          # Puppeteer Logic
├── models/
│   └── Lead.js             # Mongoose Schema
├── pages/
│   ├── _app.js             # Global Styles Import
│   ├── index.js            # Main Dashboard UI (Frontend)
│   └── api/
│       └── scrape.js       # Backend Endpoint
└── styles/
    └── globals.css         # Tailwind Imports
```

4. Step-by-Step Implementation Guide

Phase 1: Initialization & Dependencies

Action: Initialize project and install libraries.

```
npx create-next-app@latest ktym-leads --use-npm --example "[https://github.com/vercel/r
cd ktym-leads

# Install Core Logic Libraries
npm install mongoose puppeteer axios
```

```
# Install UI & Utility Libraries
npm install lucide-react clsx tailwind-merge

# Install Tailwind CSS
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Phase 2: Environment Configuration

Action: Create `.env.local` in the root. **Critical:** Do not hardcode passwords in the codebase.

```
# .env.local
MONGODB_URI=mongodb+srv://bhardwajjeet408_db_user:<YOUR_PASSWORD_HERE>@cluster0.jq6unv0
```

Phase 3: Backend Logic (The Engine)

1. Database Connection (`lib/dbConnect.js`)

Purpose: Handles caching of the MongoDB connection to prevent serverless function timeouts.

```
import mongoose from 'mongoose';

const MONGODB_URI = process.env.MONGODB_URI;

if (!MONGODB_URI) {
  throw new Error('Please define the MONGODB_URI environment variable inside .env.local
}

let cached = global.mongoose;

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null };
}

async function dbConnect() {
  if (cached.conn) return cached.conn;

  if (!cached.promise) {
    const opts = { bufferCommands: false };
    cached.promise = mongoose.connect(MONGODB_URI, opts).then((mongoose) => {
      return mongoose;
    });
  }
  cached.conn = await cached.promise;
  return cached.conn;
}

export default dbConnect;
```

2. Data Model (`models/Lead.js`)

Purpose: Defines the structure of the data saved in MongoDB.

```
import mongoose from 'mongoose';

const LeadSchema = new mongoose.Schema({
  name: { type: String, required: true },
  address: String,
  phone: String,
  website: String,
  rating: String,
  reviews: String,
  keyword: String,
  location: String,
  googleMapsLink: { type: String, unique: true }, // Prevent duplicates
  scrapedAt: { type: Date, default: Date.now },
});

// Prevent model overwrite error in dev mode
export default mongoose.models.Lead || mongoose.model('Lead', LeadSchema);
```

3. Puppeteer Scraper (`lib/scraper.js`)

Purpose: The core "Robot" that launches Chrome and scrolls Google Maps.

```
import puppeteer from 'puppeteer';

export async function scrapeGoogleMaps(keyword, location) {
  // Launch browser. Set headless: true for production, false for debugging.
  const browser = await puppeteer.launch({
    headless: false,
    args: ['--no-sandbox', '--disable-setuid-sandbox']
  });

  const page = await browser.newPage();
  await page.setViewport({ width: 1366, height: 768 });

  try {
    const query = `${keyword} in ${location}`;
    console.log(`Navigating to Google Maps for: ${query}`);

    // Navigate to Google Maps
    await page.goto(`https://www.google.com/maps/search/${query.split(" ").join("+")}`,
      waitUntil: 'networkidle2',
      timeout: 60000
    });

    // Wait for the results feed to appear
    const feedSelector = 'div[role="feed"]';
    try {
      await page.waitForSelector(feedSelector, { timeout: 10000 });
    } catch(e) {
      console.log("Could not find feed selector.");
      await browser.close();
      return [];
    }

    // Auto-scroll logic to load more results
    await page.evaluate(async (feedSelector) => {
      const wrapper = document.querySelector(feedSelector);
      await new Promise((resolve) => {
        var totalHeight = 0;
        var distance = 1000;
```

```
            var attempts = 0;
            var timer = setInterval(() => {
              var scrollHeight = wrapper.scrollHeight;
              wrapper.scrollBy(0, distance);
              totalHeight += distance;

              // Scroll until bottom or max 20 attempts
              if (totalHeight >= scrollHeight || attempts > 20) {
                clearInterval(timer);
                resolve();
              }
              attempts++;
            }, 1500);
          });
        }, feedSelector);

        // Extraction Logic
        const leads = await page.evaluate(() => {
          const items = Array.from(document.querySelectorAll('div[role="article"]'));
          return items.map(item => {
            const text = item.innerText;
            const lines = text.split('\n');
            const link = item.querySelector('a')?.href;

            return {
              name: item.getAttribute('aria-label') || lines[0],
              rating: lines.find(l => l.match(/^[0-5]\.[0-9]$/)) || 'N/A',
              reviews: lines.find(l => l.match(/\([0-9]+\)/)) || '0',
              phone: lines.find(l => l.match(/(\+\d{1,3}\s?)?\(?\d{3}\)?[\s.-]?\d{3}[\s.-]?
              address: lines.length > 2 ? lines[2] : '',
              website: '', // Requires deeper navigation to extract reliably
              googleMapsLink: link
            };
          });
        });

        await browser.close();
        return leads;

      } catch (error) {
        console.error("Scraping error:", error);
        await browser.close();
        return [];
      }
    }
```

Phase 4: API Layer

1. API Route (`pages/api/scrape.js`)

Purpose: Connects the Frontend to the Backend Scraper.

```
import dbConnect from '../../lib/dbConnect';
import Lead from '../../models/Lead';
import { scrapeGoogleMaps } from '../../lib/scraper';

export default async function handler(req, res) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method Not Allowed' });
  }
```

```
  const { keyword, location } = req.body;

  try {
    await dbConnect();

    // Run the scraper
    const leads = await scrapeGoogleMaps(keyword, location);

    if (!leads || leads.length === 0) {
      return res.status(200).json({ success: false, message: 'No leads found.' });
    }

    // Save to DB (Avoid Duplicates)
    let savedCount = 0;
    for (const lead of leads) {
      // Use Link as unique identifier
      const exists = await Lead.findOne({ googleMapsLink: lead.googleMapsLink });
      if (!exists && lead.name) {
        await Lead.create({ ...lead, keyword, location });
        savedCount++;
      }
    }

    res.status(200).json({
      success: true,
      count: leads.length,
      newlySaved: savedCount,
      data: leads
    });

  } catch (error) {
    res.status(500).json({ message: error.message });
  }
}
```

Phase 5: Frontend UI

1. Global Styles (`styles/globals.css`)

Purpose: Ensure Tailwind is loaded.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

2. Dashboard (`pages/index.js`)

Purpose: User Interface.

```
import React, { useState } from 'react';
import { Download, Search, Database, MapPin, Loader2, AlertCircle, Building2, Phone, Gl

export default function LeadScraperDashboard() {
  const [keyword, setKeyword] = useState('');
  const [location, setLocation] = useState('');
  const [isScraping, setIsScraping] = useState(false);
```

```
  const [leads, setLeads] = useState([]);
  const [logs, setLogs] = useState([]);
  const [error, setError] = useState('');

  const addLog = (message) => {
    setLogs(prev => [...prev, { time: new Date().toLocaleTimeString(), message }]);
  };

  const downloadCSV = () => {
    if (leads.length === 0) return;
    const headers = ['Name', 'Address', 'Phone', 'Rating', 'Reviews', 'Website', 'Googl
    const csvContent = [
      headers.join(','),
      ...leads.map(row => [
        `"${row.name || ''}"`, `"${row.address || ''}"`, `"${row.phone || ''}"`,
        `"${row.rating || ''}"`, `"${row.reviews || ''}"`, `"${row.website || ''}"`, `"
      ].join(','))
    ].join('\n');
    const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });
    const link = document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = `leads-${keyword}-${location}.csv`;
    link.click();
  };

  const handleStartScrape = async (e) => {
    e.preventDefault();
    if (!keyword || !location) return;
    setIsScraping(true); setError(''); setLeads([]); setLogs([]);

    addLog(`Initializing scraper for "${keyword}" in "${location}"...`);

    try {
      const response = await fetch('/api/scrape', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ keyword, location }),
      });
      const data = await response.json();

      if (data.success) {
        setLeads(data.data);
        addLog(`Success! Found ${data.count} leads.`);
        addLog(`Saved ${data.newlySaved} new unique leads to MongoDB.`);
      } else {
        setError(data.message);
        addLog(`Error: ${data.message}`);
      }
    } catch (err) {
      setError('Backend connection failed. Ensure local server is running.');
      addLog(`Critical Error: ${err.message}`);
    } finally {
      setIsScraping(false);
    }
  };

  return (
    <div className="min-h-screen bg-slate-50 text-slate-900 font-sans p-8">
      <div className="max-w-6xl mx-auto space-y-6">
        <header className="bg-white p-6 rounded-2xl shadow-sm border border-slate-200">
          <h1 className="text-2xl font-bold text-blue-700 flex items-center gap-2">
            <Database /> KTYM Lead Engine
          </h1>
        </header>
```

```
          <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
            <div className="bg-white p-6 rounded-2xl shadow-sm border border-slate-200 h-
              <h2 className="text-lg font-semibold mb-4">Target Parameters</h2>
              <form onSubmit={handleStartScrape} className="space-y-4">
                <input type="text" placeholder="e.g. Gym" value={keyword} onChange={(e) =
                <input type="text" placeholder="e.g. Patna" value={location} onChange={(e
                <button type="submit" disabled={isScraping} className="w-full bg-blue-600
                  {isScraping ? 'Processing...' : 'Start Extraction'}
                </button>
              </form>
              <div className="mt-6 bg-slate-900 rounded p-4 text-xs text-green-400 h-48 o
                {logs.map((log, i) => <div key={i}>[{log.time}] {log.message}</div>)}
              </div>
              {error && <div className="mt-4 text-red-600 text-sm">{error}</div>}
            </div>

            <div className="lg:col-span-2 bg-white rounded-2xl shadow-sm border border-sl
              <div className="flex justify-between items-center mb-4">
                <h2 className="text-lg font-bold">Results ({leads.length})</h2>
                {leads.length > 0 && <button onClick={downloadCSV} className="bg-green-60
              </div>
              <div className="overflow-auto max-h-[600px]">
                <table className="w-full text-left text-sm">
                  <thead className="bg-slate-50 sticky top-0">
                    <tr><th className="p-3">Name</th><th className="p-3">Phone</th><th cl
                  </thead>
                  <tbody>
                    {leads.map((lead, i) => (
                      <tr key={i} className="border-t hover:bg-slate-50">
                        <td className="p-3 font-medium">{lead.name}<br/><span className="
                        <td className="p-3">{lead.phone}</td>
                        <td className="p-3">{lead.rating} ({lead.reviews})</td>
                      </tr>
                    ))}
                  </tbody>
                </table>
              </div>
            </div>
          </div>
        </div>
      );
    }
```

5. Execution Instructions

1. Ensure MongoDB Atlas Cluster is running and IP is whitelisted.
2. Run `npm run dev` in the terminal.
3. Open `http://localhost:3000` .
4. Enter search terms and click "Start Extraction".