World Animals — Local DB-Driven Stack (SQLite + Express + Prisma)

This guide turns your current single-file game into a **local, database-driven** app with:

- **SQLite** database (local file)
- **Prisma** ORM + migrations
- **Express** backend API (Node.js)
- Static file hosting for your **local images** (and remote image links stored in DB)
- Frontend that **fetches animals/questions/UI strings** from the API
- Optional importer that **parses your existing** `index.html` to seed animals & questions

> Scope matches your constraints: **no CMS**, **dev-only content editing**, **EN/NL only**, **groups stay**, **images both remote & local but stored on your machine**, **no cloud**.

---

0) File tree (what you'll have at the end)

```
world-animals-db/
  .env
  package.json
  prisma/
    schema.prisma
    seed.js
  src/
    server.js
    routes/
      api.js
    lib/
      prisma.js
  scripts/
    import-country-data.js
    import-from-index.js
  data/
    country-data.json        ← copy your provided file here
  public/
    index.html               ← minimal demo UI (you can replace or integrate)
    js/
      app.js
      data-service.js
    lib/
      svg-world-map.js       ← copy your provided svg-world-map.js here
```

```
    assets/
      uploads/              ← put your local images here (served at /uploads)
```

---

1) Step-by-step install (local machine)

1) **Install Node.js (LTS ≥ 20)** - https://nodejs.org (choose LTS). On macOS you can use Homebrew: `brew install node`.

2) **Initialize the project folder**

```
mkdir world-animals-db && cd world-animals-db
```

3) **Create** `.env` (runtime settings)

```
# .env
PORT=3000
DATABASE_URL="file:./dev.db"
```

4) **Create** `package.json`

```
{
  "name": "world-animals-db",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "nodemon -w src -w prisma -x \"node --env-file=.env src/server.js\"",
    "start": "node --env-file=.env src/server.js",
    "prisma:generate": "prisma generate",
    "migrate": "prisma migrate dev --name init",
    "seed": "node prisma/seed.js",
    "import:countries": "node scripts/import-country-data.js",
    "import:index": "node scripts/import-from-index.js ./public/index.html"
  },
  "dependencies": {
    "@prisma/client": "^5.17.0",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "morgan": "^1.10.0",
    "zod": "^3.23.8"
```

```
  },
  "devDependencies": {
    "nodemon": "^3.0.1",
    "prisma": "^5.17.0"
  }
}
```

5) **Install deps**

```
npm install
```

6) **Prisma schema** → `prisma/schema.prisma`

```
// prisma/schema.prisma
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model Language {
  code  String @id   // 'en', 'nl'
  name  String
  // relations
  uiStrings   UIString[]
  animalI18n  AnimalI18n[]
  questionI18n QuestionI18n[]
  levelI18n   LevelI18n[]
}

model UIString {
  key      String
  langCode String
  value    String
  lang     Language @relation(fields: [langCode], references: [code])
  @@id([key, langCode])
  @@index([langCode])
}

model Continent {
  code String @id // 'AF','AN','AS','EU','NA','SA','OC'
```

```
  // optional i18n label via UIString keys like `continent.AF`
  countries Country[]
  animals   AnimalContinent[]
}

model Country {
  iso2          String   @id // 'NL','US', ...
  name_en       String
  name_nl       String?
  continentCode String
  continent     Continent @relation(fields: [continentCode], references: [code])
}

model Animal {
  id          Int     @id @default(autoincrement())
  slug        String  @unique // e.g. 'lion'
  groupNumber Int     // keep your current grouping semantics
  region      String?
  isActive    Boolean @default(true)
  // relations
  i18n      AnimalI18n[]
  mediaLink AnimalMedia[]
  habitats  AnimalContinent[]
  questions Question[] @relation("AnswerAnimal")
}

model AnimalI18n {
  animalId Int
  langCode String
  name        String
  description String?
  fact        String?
  altText     String?
  animal   Animal   @relation(fields: [animalId], references: [id])
  lang     Language @relation(fields: [langCode], references: [code])
  @@id([animalId, langCode])
  @@index([langCode])
}

model Media {
  id         Int    @id @default(autoincrement())
  url        String?   // remote image URL (e.g., Wikipedia)
  localPath  String?   // e.g., '/uploads/lion.jpg' (served by Express)
  role       String    // 'photo' | 'mini' | 'audio' | 'bg'
  attribution String?
  license     String?
  animals    AnimalMedia[]
}
```

```
model AnimalMedia {
  animalId Int
  mediaId  Int
  role     String?
  primary  Boolean @default(false)
  animal Animal @relation(fields: [animalId], references: [id])
  media  Media  @relation(fields: [mediaId], references: [id])
  @@id([animalId, mediaId])
  @@index([mediaId])
}

model AnimalContinent {
  animalId      Int
  continentCode String
  animal    Animal    @relation(fields: [animalId], references: [id])
  continent Continent @relation(fields: [continentCode], references: [code])
  @@id([animalId, continentCode])
  @@index([continentCode])
}

model Question {
  id            Int  @id @default(autoincrement())
  difficulty    Int  @default(1)
  groupNumber   Int?
  isActive      Boolean @default(true)
  answerAnimalId Int?
  // relations
  answerAnimal Animal? @relation("AnswerAnimal", fields: [answerAnimalId],
references: [id])
  i18n         QuestionI18n[]
}

model QuestionI18n {
  questionId Int
  langCode   String
  prompt     String
  question Question @relation(fields: [questionId], references: [id])
  lang     Language @relation(fields: [langCode], references: [code])
  @@id([questionId, langCode])
  @@index([langCode])
}

model Level {
  id          Int    @id @default(autoincrement())
  groupNumber Int
  type        String // 'explorer' | 'quiz' etc.
  goal        Int
```

```
  active      Boolean @default(true)
  i18n        LevelI18n[]
}

model LevelI18n {
  levelId  Int
  langCode String
  title       String
  description String?
  level Level     @relation(fields: [levelId], references: [id])
  lang  Language  @relation(fields: [langCode], references: [code])
  @@id([levelId, langCode])
}
```

7) **Generate client & run initial migration**

```
npx prisma generate
npm run migrate
```

8) **Seed base data** (languages, optional UI strings) → `prisma/seed.js`

```
// prisma/seed.js
import { PrismaClient } from "@prisma/client";
const prisma = new PrismaClient();

async function main() {
  // Languages EN/NL
  await prisma.language.upsert({
    where: { code: "en" },
    update: {},
    create: { code: "en", name: "English" },
  });
  await prisma.language.upsert({
    where: { code: "nl" },
    update: {},
    create: { code: "nl", name: "Nederlands" },
  });

  // Optional: a couple UI string keys as examples (add more as you move texts
into DB)
  await prisma.uIString.upsert({
    where: { key_langCode: { key: "ui.play", langCode: "en" } },
    update: { value: "Play" },
    create: { key: "ui.play", langCode: "en", value: "Play" },
```

```
  });
  await prisma.uIString.upsert({
    where: { key_langCode: { key: "ui.play", langCode: "nl" } },
    update: { value: "Spelen" },
    create: { key: "ui.play", langCode: "nl", value: "Spelen" },
  });
}

main()
  .then(async () => { await prisma.$disconnect(); })
  .catch(async (e) => { console.error(e); await prisma.$disconnect();
process.exit(1); });
```

Run it:

```
npm run seed
```

9) **Import your continents & countries** from your provided `country-data.json`

Create `data/country-data.json` (copy your file here), then add the importer as `scripts/import-country-data.js`:

```
// scripts/import-country-data.js
import { PrismaClient } from "@prisma/client";
import fs from "fs";
import path from "path";

const prisma = new PrismaClient();
const FILE = path.join(process.cwd(), "data", "country-data.json");

const CONTINENT_NAMES = {
  en: { AF: "Africa", AN: "Antarctica", AS: "Asia", EU: "Europe", NA: "North
America", SA: "South America", OC: "Oceania" },
  nl: { AF: "Afrika", AN: "Antarctica", AS: "Azië", EU: "Europa", NA: "Noord-
Amerika", SA: "Zuid-Amerika", OC: "Oceanië" }
};

async function main() {
  const raw = JSON.parse(fs.readFileSync(FILE, "utf-8"));

  // Ensure continent rows
  for (const code of Object.keys(CONTINENT_NAMES.en)) {
    await prisma.continent.upsert({ where: { code }, update: {}, create: {
code } });
```

```
    // Store i18n labels via UIString: keys like continent.AF
    for (const [langCode, names] of Object.entries(CONTINENT_NAMES)) {
      await prisma.uIString.upsert({
        where: { key_langCode: { key: `continent.${code}`, langCode } },
        update: { value: names[code] },
        create: { key: `continent.${code}`, langCode, value: names[code] },
      });
    }
  }

  // Countries (use English name from your JSON; NL name left null unless you
provide it later)
  for (const [iso2, row] of Object.entries(raw)) {
    if (iso2 === "World") continue;
    const continentCode = row.region || null; // e.g. 'EU'
    if (!continentCode) continue;
    await prisma.country.upsert({
      where: { iso2 },
      update: { name_en: row.name, continentCode },
      create: { iso2, name_en: row.name, continentCode },
    });
  }

  console.log("Imported continents & countries.");
}

main()
  .then(async () => { await prisma.$disconnect(); })
  .catch(async (e) => { console.error(e); await prisma.$disconnect();
process.exit(1); });
```

Run it:

```
npm run import:countries
```

10) **(Optional but recommended)** Import your *current* animals & questions from your provided `index.html`.

This script will **extract** `animalGroups` and `animalQuestions` from your file and insert them into the DB. It's intentionally simple and may need tiny tweaks if you change the source format.

Create `scripts/import-from-index.js`:

```
// scripts/import-from-index.js
import fs from "fs";
import path from "path";
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();
const SRC = process.argv[2] || path.join(process.cwd(), "public", "index.html");

function extractArray(namedConst, text) {
  const startToken = `const ${namedConst} = [`;
  let i = text.indexOf(startToken);
  if (i === -1) return null;
  i += startToken.length - 1; // on '['
  let depth = 0, j = i;
  for (; j < text.length; j++) {
    const ch = text[j];
    if (ch === "[") depth++;
    else if (ch === "]") { depth--; if (depth === 0) { j++; break; } }
  }
  return text.slice(i, j); // includes inner content up to ']'
}

function jsObjectToJsonish(src) {
  // Add quotes around unquoted keys: { en:"..", fact:{ nl:".." } }
  src = src.replace(/([\{,]\s*)([A-Za-z_][A-Za-z0-9_]*)\s*:/g, '$1"$2":');
  // Remove trailing commas before } or ]
  src = src.replace(/,\s*([\}\]])/g, '$1');
  return `[${src}]`; // wrap back into an array
}

async function upsertAnimal(an, groupNumber) {
  const animal = await prisma.animal.upsert({
    where: { slug: an.id },
    update: { groupNumber, region: an.region || null, isActive: true },
    create: { slug: an.id, groupNumber, region: an.region || null },
  });

  for (const [langCode, name] of Object.entries(an.name || {})) {
    await prisma.animalI18n.upsert({
      where: { animalId_langCode: { animalId: animal.id, langCode } },
      update: { name },
      create: { animalId: animal.id, langCode, name, fact: (an.fact||{})
[langCode] || null },
    });
  }

  // Media: photo + mini
```

```
  if (an.photo) {
    const m = await prisma.media.create({ data: { url: an.photo, role:
'photo' } });
    await prisma.animalMedia.create({ data: { animalId: animal.id, mediaId:
m.id, role: 'photo', primary: true } });
  }
  if (an.mini) {
    const m2 = await prisma.media.create({ data: { url: an.mini, role:
'mini' } });
    await prisma.animalMedia.create({ data: { animalId: animal.id, mediaId:
m2.id, role: 'mini' } });
  }

  // Continents
  for (const cc of (an.continents || [])) {
    try {
      await prisma.animalContinent.create({ data: { animalId: animal.id,
continentCode: cc } });
    } catch (_) { /* ignore duplicates */ }
  }
}

async function upsertQuestion(q) {
  const answerAnimal = q.answerId
    ? await prisma.animal.findUnique({ where: { slug: q.answerId } })
    : null;
  const question = await prisma.question.create({ data: { difficulty: 1,
answerAnimalId: answerAnimal?.id || null, isActive: true } });
  for (const [langCode, prompt] of Object.entries(q.prompt || {})) {
    await prisma.questionI18n.upsert({
      where: { questionId_langCode: { questionId: question.id, langCode } },
      update: { prompt },
      create: { questionId: question.id, langCode, prompt },
    });
  }
}

async function main() {
  const text = fs.readFileSync(SRC, 'utf-8');

  const groupsRaw = extractArray('animalGroups', text);
  if (!groupsRaw) throw new Error('animalGroups not found');
  const groupsJson = JSON.parse(jsObjectToJsonish(groupsRaw));

  for (let gi = 0; gi < groupsJson.length; gi++) {
    const group = groupsJson[gi];
    for (const an of group) {
      await upsertAnimal(an, gi + 1);
```

```
    }
  }

  const qRaw = extractArray('animalQuestions', text);
  if (qRaw) {
    const qJson = JSON.parse(jsObjectToJsonish(qRaw));
    for (const q of qJson) await upsertQuestion(q);
  }

  console.log('Imported animals & questions from index.html');
}

main()
  .then(async () => { await prisma.$disconnect(); })
  .catch(async (e) => { console.error(e); await prisma.$disconnect();
process.exit(1); });
```

Run it:

```
# Make sure public/index.html contains your original arrays, or pass a path to
that file
npm run import:index
```

11) **Backend code**

`src/lib/prisma.js`

```
// src/lib/prisma.js
import { PrismaClient } from "@prisma/client";
export const prisma = new PrismaClient();
export default prisma;
```

`src/routes/api.js`

```
// src/routes/api.js
import { Router } from "express";
import prisma from "../lib/prisma.js";

const router = Router();

router.get("/health", (_req, res) => res.json({ ok: true }));
```

```
router.get("/langs", async (_req, res) => {
  const langs = await prisma.language.findMany({ orderBy: { code: "asc" } });
  res.json(langs);
});

router.get("/ui", async (req, res) => {
  const lang = (req.query.lang || "en").toString();
  const rows = await prisma.uIString.findMany({ where: { langCode: lang } });
  const map = Object.fromEntries(rows.map(r => [r.key, r.value]));
  res.json(map);
});

router.get("/continents", async (req, res) => {
  const lang = (req.query.lang || "en").toString();
  const conts = await prisma.continent.findMany({ orderBy: { code: "asc" } });
  const labels = Object.fromEntries((await prisma.uIString.findMany({
    where: { langCode: lang, key: { startsWith: "continent." } }
  })).map(r => [r.key.split(".")[1], r.value]));
  res.json(conts.map(c => ({ code: c.code, name: labels[c.code] || c.code })));
});

router.get("/animals", async (req, res) => {
  const lang = (req.query.lang || "en").toString();
  const group = req.query.group ? parseInt(req.query.group) : undefined;

  const animals = await prisma.animal.findMany({
    where: { isActive: true, ...(group ? { groupNumber: group } : {}) },
    orderBy: [{ groupNumber: "asc" }, { slug: "asc" }],
    include: {
      i18n: { where: { langCode: lang } },
      mediaLink: { include: { media: true } },
      habitats: { include: { continent: true } },
    },
  });

  const shaped = animals.map(a => ({
    id: a.slug,
    groupNumber: a.groupNumber,
    region: a.region,
    name: a.i18n[0]?.name || a.slug,
    fact: a.i18n[0]?.fact || null,
    photos: a.mediaLink
      .filter(m => m.media.role === 'photo')
      .map(m => m.media.url || m.media.localPath),
    mini: a.mediaLink
      .filter(m => m.media.role === 'mini')
      .map(m => m.media.url || m.media.localPath)[0] || null,
    continents: a.habitats.map(h => h.continent.code),
```

```
  }));

  res.json(shaped);
});

router.get("/animals/:slug", async (req, res) => {
  const lang = (req.query.lang || "en").toString();
  const { slug } = req.params;
  const a = await prisma.animal.findUnique({
    where: { slug },
    include: {
      i18n: { where: { langCode: lang } },
      mediaLink: { include: { media: true } },
      habitats: { include: { continent: true } },
    },
  });
  if (!a) return res.status(404).json({ error: "Not found" });
  res.json({
    id: a.slug,
    groupNumber: a.groupNumber,
    region: a.region,
    name: a.i18n[0]?.name || a.slug,
    fact: a.i18n[0]?.fact || null,
    photos: a.mediaLink.filter(m => m.media.role === 'photo').map(m =>
m.media.url || m.media.localPath),
    mini: a.mediaLink.filter(m => m.media.role === 'mini').map(m => m.media.url
|| m.media.localPath)[0] || null,
    continents: a.habitats.map(h => h.continent.code),
  });
});

router.get("/questions/random", async (req, res) => {
  const lang = (req.query.lang || "en").toString();
  const difficulty = req.query.difficulty ? parseInt(req.query.difficulty) :
undefined;
  const group = req.query.group ? parseInt(req.query.group) : undefined;

  const where = { isActive: true, ...(difficulty ? { difficulty } : {}), ...
(group ? { groupNumber: group } : {}) };
  const total = await prisma.question.count({ where });
  if (!total) return res.status(404).json({ error: "No questions" });
  const skip = Math.floor(Math.random() * total);
  const q = await prisma.question.findFirst({ where, skip, include: { i18n: {
where: { langCode: lang } }, answerAnimal: true } });
  res.json({ id: q.id, prompt: q.i18n[0]?.prompt || "", answerId:
q.answerAnimal?.slug || null });
});
```

```
export default router;
```

`src/server.js`

```
// src/server.js
import express from "express";
import path from "path";
import cors from "cors";
import morgan from "morgan";
import api from "./routes/api.js";
import dotenv from "dotenv";

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

// Serve local images at /uploads/*
app.use("/uploads", express.static(path.join(process.cwd(), "public", "assets",
"uploads")));

// API
app.use("/api", api);

// Frontend (static demo)
app.use(express.static(path.join(process.cwd(), "public")));

const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Server listening on http://localhost:${port}
`));
```

12) **Frontend (minimal demo)**

This is a **lightweight demo UI** to prove the API + DB flow. You can merge these patterns into your full game. It fetches animals by group and one random question, and shows how to use local images via `/uploads/...`.

`public/index.html`

```
<!doctype html>
<html lang="en">
```

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>World Animals — DB Demo</title>
  <style>
    body { font-family: system-ui, Arial, sans-serif; margin: 0; padding:
20px; }
    header { display:flex; gap:12px; align-items:center; }
    .row { display:flex; flex-wrap:wrap; gap:14px; margin: 16px 0; }
    .card { border:1px solid #ddd; border-radius:12px; padding:12px; width:
220px; }
    .card img { width: 100%; height: 140px; object-fit: cover; border-radius:
8px; }
    .muted { color:#666; font-size: 12px; }
  </style>
</head>
<body>
  <header>
    <h1>World Animals — DB Demo</h1>
    <select id="lang">
      <option value="en">English</option>
      <option value="nl">Nederlands</option>
    </select>
    <label>Group <input id="group" value="1" size="2" /></label>
    <button id="load">Load</button>
  </header>

  <section>
    <h2>Animals</h2>
    <div id="animals" class="row"></div>
  </section>

  <section>
    <h2>Random Question</h2>
    <button id="questionBtn">Get Question</button>
    <div id="question"></div>
  </section>

  <!-- Optional: your map library if you want to render a map later -->
  <!-- <script src="/lib/svg-world-map.js"></script> -->
  <script src="/js/data-service.js" type="module"></script>
  <script src="/js/app.js" type="module"></script>
</body>
</html>
```

`public/js/data-service.js`

```
// public/js/data-service.js
const API = "/api";

export async function getAnimals({ lang = "en", group = 1 } = {}) {
  const url = new URL(`${API}/animals`, location.origin);
  url.searchParams.set("lang", lang);
  url.searchParams.set("group", group);
  const res = await fetch(url);
  if (!res.ok) throw new Error("Failed to fetch animals");
  return res.json();
}

export async function getRandomQuestion({ lang = "en", group, difficulty } =
{}) {
  const url = new URL(`${API}/questions/random`, location.origin);
  url.searchParams.set("lang", lang);
  if (group) url.searchParams.set("group", group);
  if (difficulty) url.searchParams.set("difficulty", difficulty);
  const res = await fetch(url);
  if (!res.ok) throw new Error("No question available");
  return res.json();
}

export async function getContinents({ lang = "en" } = {}) {
  const url = new URL(`${API}/continents`, location.origin);
  url.searchParams.set("lang", lang);
  const res = await fetch(url);
  if (!res.ok) throw new Error("Failed to fetch continents");
  return res.json();
}

export async function getUIStrings({ lang = "en" } = {}) {
  const url = new URL(`${API}/ui`, location.origin);
  url.searchParams.set("lang", lang);
  const res = await fetch(url);
  if (!res.ok) return {};
  return res.json();
}
```

`public/js/app.js`

```
// public/js/app.js
import { getAnimals, getRandomQuestion } from "./data-service.js";

const $animals = document.getElementById("animals");
```

```
const $lang = document.getElementById("lang");
const $group = document.getElementById("group");
const $load = document.getElementById("load");
const $questionBtn = document.getElementById("questionBtn");
const $question = document.getElementById("question");

async function renderAnimals() {
  const lang = $lang.value; const group = parseInt($group.value || "1");
  const animals = await getAnimals({ lang, group });
  $animals.innerHTML = animals.map(a => `
    <div class="card">
      ${a.mini ? `<img src="${a.mini}" alt="${a.name}"/>` : ''}
      <h3>${a.name}</h3>
      ${a.fact ? `<p class="muted">${a.fact}</p>` : ''}
      <p class="muted">Continents: ${a.continents.join(', ')}</p>
    </div>
  `).join("");
}

async function renderQuestion() {
  const lang = $lang.value; const group = parseInt($group.value || "1");
  try {
    const q = await getRandomQuestion({ lang, group });
    $question.textContent = q.prompt || "";
  } catch (e) {
    $question.textContent = "No question available.";
  }
}

$load.addEventListener("click", renderAnimals);
$questionBtn.addEventListener("click", renderQuestion);

// initial load
renderAnimals();
```

13) **Put your files in place**

- Copy your provided `svg-world-map.js` into `public/lib/svg-world-map.js` (so you can keep using the same map library when you integrate it into the demo or your full UI).
- Copy your provided `country-data.json` into `data/country-data.json` and run `npm run import:countries`.
- (Optional) Put your **local images** into `public/assets/uploads/` — they'll be available at `http://localhost:3000/uploads/<filename>` and you can create `Media` rows with `localPath: "/uploads/<filename>"`.
- (Optional, recommended) Run `npm run import:index` to ingest `animalGroups` and `animalQuestions` from your existing HTML.

14) **Run the app**

```
npm run dev
# open http://localhost:3000
```

---

2) What else to do to make the game “workable” end-to-end

- **Move UI text into DB** progressively: for each hard-coded label in your game, add a `UIString` row for `en`/`nl` and read it via `/api/ui?lang=...`. Start with top-level labels (Play, New Game, Missions, etc.).
- **Keep groups as numbers**: your current `animalGroups[0..n]` becomes `Animal.groupNumber = 1..n`. Use `/api/animals?group=1` to fetch per group.
- **Add more media**: store multiple `Media` rows per animal (`photo` as main image, `mini` as thumbnail). For local images, drop files into `/public/assets/uploads` and store `localPath` in DB. For remote images, store the remote `url`.
- **Balance without redeploy**: if you later move your mission/level thresholds into the `Level` table, you can adjust values in the DB and have the frontend respect them from `/api/levels` (endpoint trivial to add following the patterns above).
- **Persist player progress locally** (unchanged from your current approach) using `localStorage` on the client; only introduce server-side profiles later if needed.
- **Back up your DB file**: your entire data is in `dev.db`. Back it up regularly (copy the file) or version it.
- **Deploy later**: when you’re ready for the internet, you can host the same code on any VPS or PaaS. SQLite works fine for small traffic; moving to Postgres later is a 1–2 hour Prisma migration.

---

3) Quick checklist

- [ ] Node installed (v20+)
- [ ] `.env` created with `DATABASE_URL=file:./dev.db`
- [ ] `npm install`
- [ ] Prisma generated + migrated
- [ ] Seed run (`npm run seed`)
- [ ] Countries imported (`npm run import:countries`)
- [ ] (Optional) Animals & questions imported from your `index.html` (`npm run import:index`)
- [ ] Local images placed under `public/assets/uploads/`
- [ ] `npm run dev` and open `http://localhost:3000`

Notes

- The `import-from-index.js` script is a pragmatic parser tailored to your current file. If you later restructure your source arrays, adjust the regex helpers accordingly.
- All **links to pictures** (remote or local) are **stored in the DB** (`Media.url` or `Media.localPath`).

- All **texts and descriptions** are in the DB (`AnimalI18n`, `QuestionI18n`, `UIString`).
- Adding **levels/animals/text** becomes: insert DB rows → frontend automatically renders via the API.

If you want, I can also add a lightweight POST endpoint + CLI to add/edit animals and media locally (dev-only), or wire up a tiny admin HTML form (no CMS, just your machine).