Smart Sagar â Project Summary

Overview
Smart Sagar is a marine operations dashboard designed to support fishing crews and coastal operations. It combines a modern React + Vite application shell with legacy HTML/JS tools, allowing older dashboards to run inside the new singleâ page app. The goal is to provide a unified control deck for safety, weather, fish detection, and biodiversity intelligence without rewriting all legacy tools at once.

Problems It Solves
- Operational awareness: Consolidates weather, safety, fish population, and map intelligence in one place.
- Faster decisionâ making: Provides quick recommendations, alerts, and risk summaries.
- Legacy preservation: Keeps existing HTML tools working while adding new React screens.
- Field usability: Lightweight clientâ side UI that runs as a static web app.

Core User Workflows
- Main dashboard: View operational status, weather, trip planning, and AI copilot quick actions.
- Fish detect: Type a species name, autoâ complete suggestions appear, predict population trend and extinction risk, and visualize habitats on a map.
- Marine weather: Interactive map with clickâ toâ inspect weather and risk summaries; export and comparison tools.
- SOS emergency: Create an incident profile, share location, find nearest coastal station, and trigger distress protocols.
- Legacy biodiversity explorer: Fetch biodiversity occurrence data from public sources and render in map/analytics views.

Highâ Level Architecture
- React app shell (Vite + React Router) provides navigation and layout.
- React pages either render UI directly (Home, SOS, Fish Detect, Weather) or load legacy HTML tools inside a wrapper.
- Legacy tools live in public/legacy and are fetched at runtime by the React app.

How It Works (Detailed)
1) App shell and routing
- src/App.jsx wires routes with react-router-dom.
- Global overlays and nav live in src/components (e.g., Fish3DOverlay, GlobalNavbar).
- Each route is either a React page or a LegacyPage wrapper.

2) React pages with legacy assets
- React pages (Home, SOS, Weather, Fish Detect) call usePageAssets.
- usePageAssets injects legacy CSS/JS assets and also attaches a shared ui-theme.css for consistent styling.
- This lets the page behave like the legacy HTML version while being rendered inside React.

3) Legacy page hydration
- src/LegacyPage.jsx fetches the legacy HTML from /public/legacy via /legacy/* routes.
- It parses HTML, injects link/script/style tags into the current document, rewrites asset URLs, and loads scripts sequentially.
- It also dispatches DOMContentLoaded and load events so existing scripts run without changes.
- Internal legacy links are intercepted and routed through React Router.

Key Features by Page

- Home dashboard (src/pages/HomePage.jsx)
  - Loads legacy styles and scripts, plus Google Translate.
  - Uses browser geolocation and fetches live weather from Openâ Meteo API.
  - Shows a synthetic â AI Marine Copilotâ panel with quick actions; uses Chatbase embed for optional chat.

- Fish Detect (src/pages/FishDetectPage.jsx + public/legacy/fish_detect/script.js)
  - Autocomplete for fish species and prediction model for population trend and extinction risk.
  - Uses Leaflet for habitat map and markers.
  - Local fish database is stored in JS and used to generate predictions; export to JSON is available.

- Weather (src/pages/WeatherPage.jsx + public/legacy/wether/script.js)
  - Interactive Leaflet map; click locations to fetch Openâ Meteo forecasts.
  - Displays storm alerts, risk intelligence, comparisons, and export options.

- SOS (src/pages/SosPage.jsx + public/legacy/sos/sos-emergency.js)
  - Emergency incident profile, crew and severity tracking, location sharing, nearest station lookup.
  - UX optimized for rapid response.

- Legacy biodiversity explorer (public/legacy/dashoardfinal/index.html)
  - Uses GBIF and OBIS public APIs for occurrence data and biodiversity insights.

Tech Stack
- Frontend: React 18, React Router DOM 6, Vite 5
- Mapping: Leaflet + OpenStreetMap tiles (legacy pages)
- Styling: Custom CSS + shared ui-theme.css
- Thirdâ party services: Openâ Meteo (weather), Chatbase (optional chat), Google Translate
- Legacy assets: HTML/CSS/JS served from public/legacy

Backend / Data Layer (Current State)
- There is no dedicated backend in this repository.
- All logic runs clientâ side in the browser.
- Data sources are public APIs (Openâ Meteo, GBIF, OBIS) or local static JSON/JS in public/legacy.
- Persistence is not implemented; session data is in memory unless exported (e.g., Fish Detect export JSON).

Runtime Data Flow (Example: Fish Detect)
1) User types a species name.
2) Autocomplete suggestions come from the local fish database in public/legacy/fish_detect/script.js.
3) Prediction logic runs clientâ side and updates UI cards.
4) Habitat locations are plotted on Leaflet map; markers are colored by risk status.
5) User can export the prediction report as JSON.

Why This Architecture
- Allows rapid delivery of new React pages while preserving existing tools.
- Minimizes risk by keeping legacy behaviors intact.
- Provides a single deployment artifact (static assets + Vite build output).

Known Gaps / Next Steps (If Needed)
- Add a real backend for user accounts, persistence, audit logs, and analytics.

- Centralize data ingestion and caching to reduce API calls from the client.
- Introduce authentication and roleâ based access for operational systems.