**Ship Tracking and Alerting Application**

**1. Project Overview**

This document provides a technical overview of the Ship Tracking and Alerting Application. This is a MERN Stack application designed to monitor vessel locations and send real-time alerts to multiple designated engineers via WhatsApp based on proximity to a selected port.

The application ensures continuity by providing a reliable and clear structure for future maintenance and development.

**1.1 Key Workflow**

1. **Input:** A user enters a Vessel Name, selects one or more Engineers for notification, and chooses a Port to track the vessel against.
2. **Tracking:** A recurring background process (Cron/Workers) uses external APIs to retrieve the vessel's location data.
3. **Alerting:** The system calculates the Estimated Time of Arrival (ETA) and proximity, triggering alerts to the selected engineers via WhatsApp upon specific events. Frontend alerts are managed via Notistack.

**2. Technical Stack and Environment**

| **Category** | **Technology/Tool** | **Details** |
| --- | --- | --- |
| **Frontend** | React, Redux, Tailwind CSS, Notistack | React for UI, Redux for state, Tailwind for utility-first styling, Notistack for transient frontend notifications. |
| **Backend** | Node.js, Express, MongoDB | Core backend framework and database. |
| **Notifications** | Twilio (WhatsApp API) | Used for sending external alerts to engineers. |
| **Server/Deployment** | Webmin | The application is deployed and managed on a server utilizing the Webmin control panel. |
| **Live Schedule** | PM (24/7) | The server is configured to be live and operational 24 hours a day, 7 days a week (PM). |

**3. Backend Architecture (Node.js/Express)**

The backend is structured to manage data persistence, external API calls, business logic, and notifications.

**3.1 Backend Directory Structure**

| **Folder/File** | **Purpose** | **Note** |
| --- | --- | --- |
| **Auth Info** | Authentication utility created by Bailey's for WhatsApp notification security. | *Ignore for standard development.* |
| **Config** | Contains static configuration files for runtime behavior. |  |
| **Cron** | Entry point for initiating the recurring scheduled jobs. | Calls functions defined in the Worker folder. |
| **DB** | Database connection and configuration logic. | Mongoose connection setup. |
| **Source** | Contains the main application business logic. |  |

**3.2 Configuration Details (Config Folder)**

| **File** | **Responsibility** |
| --- | --- |
| **NotificationConfig** | Defines the templates, triggers, and content for all notifications sent to engineers. |
| **PortAliases** | Crucial mapping logic. Used to normalize port destination data received from external APIs (which may return port names, URLs, or other sources) into a standard format for comparison and tracking. |

**3.3 Core Application Logic (Source Folder)**

The Source folder is the heart of the application, managing all data flow and business rules.

**Controllers**

The controllers handle incoming HTTP requests and coordinate responses.

| **Controller** | **Responsibility** |
| --- | --- |
| **Engineer Controller** | CRUD operations for Engineer profiles (name, WhatsApp number). |
| **Port Controller** | CRUD operations for the ports available for tracking. |
| **Ship Controller** | Handles general ship data unrelated to active tracking. |
| **Vessel Controller** | CRUD operations for currently tracked vessels (delete, update, retrieval of tracked list). |
| **Vessel Finder Controller** | Manages the sequence and logic for calling the external APIs (Vessel Finder API and AIS Hub API). |

**Models, Routes, and Workers**

| **Folder** | **Key Responsibilities** | **Files Contained (Examples)** |
| --- | --- | --- |
| **Model** | Defines the data schema (Mongoose schemas). | Engineer Model, Port Model, Ship Model, Vessel Model. |
| **Route** | Defines the API endpoints. | Engineer Route, Port Route, Vessel Route, Vessel Finder Route. |
| **Worker** | Executes the Cron jobs and background processes. | AIS Worker (handles AIS/Vessel Finder polling), Notification Worker (processes and dispatches alerts). |

**Services and Utils**

| **Folder** | **Responsibility** | **Files Contained (Examples)** |
| --- | --- | --- |
| **Services** | Encapsulates the core business logic. | ETA Service, Notification Service, Queue, Vessel, WhatsApp. |
| **Utils** | Contains reusable, stateless helper functions. | ETA, Formatter, Matcher (for port alias comparison), Vessel Utils. |

**3.4 Detailed API Polling Strategy and Cost Management**

The application employs a Two-Tiered API Polling Strategy managed by the logic within the Vessel Finder Controller and executed by the Worker processes. The goal is to maximize the use of the free AIS Hub API and strictly limit calls to the paid Vessel Finder API to control operational costs.

**A. Tier 1: AIS Hub API (Free Tier)**

The system prioritizes the free AIS Hub API. Its standard use is governed by its native rate limits (often limited to once per minute to receive data, as per standard documentation).

| **Step** | **Action** | **Outcome** |
| --- | --- | --- |
| **1. Initial Call** | The Cron job (Worker) first attempts to retrieve the vessel's position and destination using the AIS Hub API. | Success: Data is processed and used to update the vessel's position and ETA. |
| **2. Failure / Missing Data** | If the AIS Hub API call fails (due to AISHUB API doesn’t contain the data about that specific vessel), the system executes a Fallback Mechanism. | Failure: The system immediately transitions to the Vessel Finder API (Tier 2). |

**B. Tier 2: Vessel Finder API (Paid Tier - High Cost Control)**

The paid Vessel Finder API is only called when Tier 1 is unsuccessful, and its use is governed by a strict, time-based rate limit tied directly to the vessel's tracking status to manage expense.

The system uses the vessel's reported destination (from any available data) and compares it to the Target Port defined by the user in the system.

| **Condition** | **Logic/Comparison** | **API Call Frequency to Vessel Finder** | **Rationale (Cost Management)** |
| --- | --- | --- | --- |
| **Destination Match** | The vessel's reported DEST (Destination from AIS data) matches the configured Target Port (using the normalization logic in PortAliases). | Once Every 6 Hours | The vessel is confirmed to be on the correct track, requiring less frequent checks to monitor for final proximity alerts. |
| **Destination Mismatch** | The vessel's reported DEST does not match the configured Target Port. | Once Every 24 Hours | The vessel is likely not proceeding to the target port. Expensive API calls are significantly minimized, as the data is less relevant until the destination changes. |

**C. Polling State Flow (Controller Logic)**

The logic flow within the Vessel Finder Controller ensures continuous, cost-optimized tracking:

1. **Iterate:** The system iterates over all active vessels in the Vessel Model.
2. **Check AIS Hub:** Call the AIS Hub API.
3. **Process Result:**
   * If AIS Hub data is valid and sufficient, update the vessel record and calculate ETA. STOP (Do not call Vessel Finder).
   * If AIS Hub data is invalid/missing, proceed to the Vessel Finder check.
4. **Rate Limit Check:** For the current vessel, check the last successful call time to the Vessel Finder API against the required 6-hour or 24-hour interval (based on the Destination Match status).
5. **Execute Vessel Finder Call:**
   * If the required time limit has elapsed, call the Vessel Finder API, update the vessel data, and record the new timestamp of the call.
   * If the required time limit has NOT elapsed, SKIP the call to Vessel Finder and proceed to the next vessel, effectively conserving API credits.

This detailed, condition-based logic ensures the application provides the necessary reliability without incurring unnecessary costs from the paid data source.

**4. Frontend Architecture (React/Redux) - Detailed Breakdown**

The frontend is a **Single Page Application (SPA)** built with **React**, utilizing **Redux** for state management, and styled with **Tailwind CSS**. It provides an intuitive interface for monitoring vessels and managing notification settings.

**4.1 State Management (Redux)**

* **Redux Implementation:** Redux is used for global state management.
* **Focus:** The Redux store is currently implemented **only for Ship/Vessel-related data**. This centralizes tracking information, making it easily accessible across the Monitoring page, the Ship Detail page, and various components without prop-drilling.

**4.2 Frontend Folder Structure**

| **Folder** | **Purpose** |
| --- | --- |
| **Components** | Houses all reusable UI and page-specific elements. |
| **Hooks** | Contains custom React hooks for encapsulating and reusing complex logic. |
| **Pages** | Defines the top-level container components for each application route. |
| **Redux** | Contains the store configuration, reducers, and actions (currently focused on Vessel data). |
| **Services** | Handles communication with the backend API. |
| **Utils** | Contains reusable utility functions for data manipulation. |

**4.3 Components Directory (Components)**

The components are logically grouped into folders based on the page they primarily serve, plus a folder for universally shared components.

| **Component Folder** | **Primary Purpose** | **Examples of Components** |
| --- | --- | --- |
| **AddVessel** | Houses components specific to the vessel tracking input form. | VesselNameInput, EngineerSelect (Multi-select), PortDropdown, etc. |
| **Common** | Contains highly reusable UI elements shared across multiple pages. | Header, Footer, Navigation Bar, Generic Button, Modal, Loading Spinner. |
| **Engineer** | Houses components specific to the Engineer Form page. | EngineerTable, AddEngineerForm, EngineerDetailsCard. |
| **Monitoring** | Houses components for the dashboard overview. | VesselTrackingTable, PaginationControls, DeleteVesselButton. |
| **ShipDetails** | Houses components for the single-vessel view. | VesselInfoCard, StatusHistoryChart. |
| **ShipMap** | Contains all map-related logic and display components. | MapView, VesselMarker, RouteDisplay. |
| **Protected Route** | A higher-order component (HOC) or functional component wrapper used to enforce the password security gate on application routes. | Enforces authentication before rendering child routes. |

**4.4 Hooks Directory (Hooks)**

Custom hooks abstract complex state and side-effect logic, making components cleaner and business logic reusable.

| **Hook Name** | **Responsibility** |
| --- | --- |
| **useAddVessel** | Manages the form state, validation, and submission logic for the **Add Vessel Page**. |
| **useAISData** | Handles the logic for fetching and managing real-time vessel position data from the backend. |
| **usePorts** | Logic for fetching and managing the list of available ports from the backend for selection fields. |
| **useVesselData** | Retrieves and maintains the data for a **single vessel** (used primarily on the Ship Details Page). |
| **useVessels** | Retrieves and maintains the state for **all tracked vessels** (used primarily on the Monitoring Page). |

**4.5 Pages Directory (Pages)**

The frontend is structured into five main pages, each serving a distinct purpose in the vessel tracking workflow.

| **Page Name** | **Functionality and Workflow Details** |
| --- | --- |
| **Password** | **Security Gateway:** The initial route that enforces access control by requiring a hardcoded password. Access to all other routes is blocked until successful authentication. |
| **EngineerForm** | **Engineer Management:** A dedicated page containing a form with three fields: **Engineer Name, Email, and Phone Number**. It also displays a comprehensive list (table) of all existing engineer profiles. |
| **AddVessel** | **Tracking Initiation Workflow:** This is the core data input page, featuring: 1. **Vessel Lookup:** As the user types the ship name, the system attempts to auto-complete the name and fill the **MMSI code** using existing backend data. 2. **New Vessel Creation:** If the vessel is *not* found in the backend database, the system allows the user to manually enter the Name and MMSI. Upon clicking **Add Vessel**, the system will **first persist the new ship details to the database** and then automatically start the tracking job. 3. **Input Fields:** Users select the **Engineers** (supports multiple selections) to receive alerts and the **Target Port**. 4. **Navigation:** Upon successful addition, the user is **automatically navigated to the ShipDetails page** for the newly added vessel. |
| **ShipDetails** | **Single Vessel View:** Dedicated view showing only the actively tracked vessel that was just added or selected. **UI Details:** Displays a **Map** highlighting the vessel's current position and route. It includes a detailed **Table/Field Section** showing all critical real-time information (e.g., Navigation Status, Latitude, Longitude, SOG, COG). |
| **Monitoring** | **Tracking Dashboard and Lifecycle Management:** The central hub for all tracking activities. **UI Details:** Displays a **Map** that shows the real-time location of **ALL** actively tracked vessels. **Tracking Filter/Lifecycle:** Features two primary tabs or filters: **Tracking** (for actively monitored vessels) and **Completed** (for vessels that have completed their alert cycle/arrived). **Data Deletion Policy:** Vessels moved to the **Completed** section are automatically **deleted from the database after a 15-day grace period** to manage data storage and maintain relevance. |

**4.6 Services and Utilities**

This section details the non-component, non-hook files essential for data management and backend communication within the frontend application.

| **Folder/File** | **Detailed Responsibility** |
| --- | --- |
| **Services/API.js** | Backend Communication Layer (Data Access): This centralized file contains all functions responsible for making asynchronous HTTP requests to the backend routes (e.g., api.getVessels(), api.addEngineer(), api.trackVessel()). It isolates all network logic from the components and hooks. |
| **Utils/Format** | Data Formatting Utility: Contains helper functions for transforming raw data received from the backend into a user-friendly format for display, such as standardizing date/time formats, converting distance units, and rounding numbers. |
| **Utils/ShipName** | Ship Name Handling Logic: Contains specialized functions for sanitizing, validating, or formatting vessel/ship names. This is critical for ensuring consistent query behavior and clean display across the application. |
| **Utils (Helper Functions)** | General Purpose Utilities: Houses various other stateless utility functions used throughout the frontend codebase (e.g., array manipulation, simple calculations, input validation helpers). |

**4.7 Security Note (Password Protection)**

The application utilizes a **password gate** implemented via the Password page and the Protected Route component. The password is **hardcoded** within the frontend code, which provides a basic layer of protection but is **not suitable for production environments requiring high security** (e.g., it is visible in the source code). This should be noted for any future security audits or upgrades.