**Holy Crosser Database Administrator – Rapscallion Build - Functional Requirements Document**

Application Overview

Database administration system for managing Holy Island crossing data with automated batch updates and manual data management capabilities.

**It uses an Existing Database Schema (PostgreSQL)**

The application connects to an existing database with these tables:

- crossing\_times (contains your historical crossing data)

- tide\_data (contains collected tide information)

- weather\_data (contains weather history)

- api\_settings (contains API configuration)

Database Schema (PostgreSQL)

-- Crossing times data

TABLE crossing\_times (

id SERIAL PRIMARY KEY,

date DATE NOT NULL,

safe\_crossing\_time TIME,

next\_safe\_time TIME,

tide\_height DECIMAL(4,2),

weather\_conditions VARCHAR(255),

source\_url TEXT,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Tide data

TABLE tide\_data (

id SERIAL PRIMARY KEY,

datetime TIMESTAMP NOT NULL,

height DECIMAL(5,2) NOT NULL,

tide\_type VARCHAR(50),

location VARCHAR(255) DEFAULT 'Holy Island',

api\_source VARCHAR(100),

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Weather data

TABLE weather\_data (

id SERIAL PRIMARY KEY,

datetime TIMESTAMP NOT NULL,

temperature DECIMAL(4,1),

humidity INTEGER,

wind\_speed DECIMAL(4,1),

wind\_direction VARCHAR(10),

conditions VARCHAR(255),

forecast\_day INTEGER,

api\_source VARCHAR(100),

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- API settings

TABLE api\_settings (

id SERIAL PRIMARY KEY,

data\_type VARCHAR(50) NOT NULL,

source\_name VARCHAR(255) NOT NULL,

base\_url TEXT NOT NULL,

api\_key\_required BOOLEAN DEFAULT false,

update\_frequency VARCHAR(50),

is\_active BOOLEAN DEFAULT true,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

Environment Variables Required

* DATABASE\_URL - PostgreSQL connection string
* UKHO\_API\_KEY - UK Hydrographic Office API key for tide data
* OPENWEATHERMAP\_API\_KEY - Weather API key (if using OpenWeatherMap)
* GMAIL\_USER - Gmail address for notifications
* GMAIL\_APP\_PASSWORD - Gmail app password for SMTP
* ADMIN\_EMAIL - Email address to receive automation reports

Core Functionality Requirements

1. Admin Dashboard (3 Tabs)

**Scripts Tab:**

* Display status of Weather Update and Tide Data Update scripts
* Show last run time, next scheduled run, records processed
* Buttons for "Test Run" (dry run) and "Execute" (live run)
* Visual status indicators (success/error/running)

**Schedule Tab:**

* Start/Stop automation system controls
* Display current automation status (Running/Stopped)
* Show cron schedules: Weather (06:15, 12:15, 18:15 UK time), Tides (03:15 UK time)
* Transition states during start/stop operations

**Settings Tab:**

* Dry run mode toggle (global setting for test executions)
* Email test functionality
* System health indicators

2. Data Management Interface

**CRUD Operations:**

* View paginated data tables for crossing times, tide data, weather data
* Add new records manually with form validation
* Edit existing records with pre-populated forms
* Delete records with confirmation
* Search and filter capabilities
* Bulk data download functionality

3. Automation System

**Cron Scheduling:**

* Weather updates: 3 times daily (06:15, 12:15, 18:15 UK time)
* Tide data updates: Daily at 03:15 UK time
* Automatic failover and error handling
* Email notifications for script completion/errors

**Script Execution:**

* Manual trigger capability with dry-run option
* Progress tracking and status reporting
* Database backup before bulk operations
* Transaction rollback on errors

4. API Integrations

**Weather Data Sources:**

* Open-Meteo API (free, no key required)
* OpenWeatherMap API (backup option)
* 7-day forecast data collection

**Tide Data Sources:**

* UK Hydrographic Office API (requires UKHO\_API\_KEY)
* Tide predictions for Holy Island area

**Crossing Times Sources:**

* Northumberland County Council website scraping
* HTML parsing for crossing time tables

5. Authentication & Security

* Simple username/password authentication
* Session management
* Admin-only access (single user system)
* API endpoint protection

6. Data Export/Import

**Query Builder:**

* Date range selection
* Data type filtering (crossing times, tides, weather)
* Export formats: JSON, CSV
* Bulk download functionality

7. System Health Monitoring

**Health Checks:**

* Database connectivity
* Automation system status
* External API availability
* Email system functionality

8. Email Notifications

**Automated Reports:**

* Script execution summaries
* Error notifications
* Daily status reports
* Test email functionality

API Endpoints Required

GET /api/admin/scripts/status - Script status information

POST /api/admin/scripts/{name}/run - Execute script manually

GET /api/admin/system/health - System health check

GET/POST /api/admin/automation/start - Start automation

GET/POST /api/admin/automation/stop - Stop automation

GET /api/admin/automation/status - Automation status

GET /api/crossing-times - Paginated crossing times data

POST /api/crossing-times - Create new crossing time

PUT /api/crossing-times/{id} - Update crossing time

DELETE /api/crossing-times/{id} - Delete crossing time

GET /api/tide-data - Paginated tide data

POST /api/tide-data - Create new tide data

PUT /api/tide-data/{id} - Update tide data

DELETE /api/tide-data/{id} - Delete tide data

GET /api/weather-data - Paginated weather data

POST /api/weather-data - Create new weather data

PUT /api/weather-data/{id} - Update weather data

DELETE /api/weather-data/{id} - Delete weather data

POST /api/query-external - Manual API data fetching

POST /api/admin/test-email - Send test email

Key Features

* UK timezone handling (BST/GMT conversion)
* Mobile-responsive design
* Real-time status updates
* Error handling and recovery
* Data validation and sanitization
* Automated backup and safety measures

This specification provides everything needed to rebuild the application with any frontend technology while maintaining all current functionality and avoiding the caching issues we've experienced.

**Front end technology**

As this is a admin tool it doesn’t have to be beautiful, just functional. The key thing is that it is as simple as possible so I don’t waste time on front end cacheing issues etc.

We would like to use a Python Flask app with simple.

Please adhere to these rule as it will also be used on a mobile phone:

**Mobile Best Practices for Flask:**

* Large tap targets (44px minimum)
* Simple navigation (hamburger menu)
* Form inputs optimized for mobile keyboards
* Minimal JavaScript (only for essential interactions)
* Fast server responses (which your backend already provides)