

--- FILE: ImplementationPlan.md ---

Implementation Plan - Phase 3: UI Implementation

Version: 1.1.1 **Target:** 0.103.0-Beta

1. Executive Summary

This plan executes the **UI Implementation** phase of the Web Pathfinder. It establishes the Django application structure (**analyzer** app), implements the "Three-Slot" upload interface, and connects the web layer to the core **LogManager** logic.

2. Architecture & Constraints

- **Statelessness:** Files are handled via `tempfile` and processed immediately. No permanent storage.
- **Identity Agnostic:** Labels are "Log 1", "Log 2", "Log 3" (No "My Log").
- **Shared Presentation:** Uses the existing `contest_tools` logic for parsing.
- **UI Framework:** Bootstrap 5 (via CDN) for layout, consistent with existing report styles.

3. Proposed Changes

3.1. Configuration Updates

- `web_app/config/settings.py`: Register the new `web_app.analyzer` app.
- `web_app/config/urls.py`: Route the root URL `""` to the analyzer app.

3.2. New Application (`web_app/analyzer/`)

- `apps.py`: App configuration.
- `forms.py`: `UploadLogForm` with three fields (Log 1 required, 2 & 3 optional).
- `views.py`:
 - `home`: Renders the upload form.
 - `upload`: Handles POST, creates temp dir, runs **LogManager**, and renders the dashboard.
- `urls.py`: App-specific routing.

3.3. Templates

- **base.html:** Common HTML skeleton (Bootstrap 5 CDN, Navbar, Footer).
- **home.html:** The Upload Form (Three Slots).
 - **Labeling:** "Log 1", "Log 2", "Log 3".
 - **Instruction:** "Select Cabrillo File".
- **dashboard.html:** A skeletal results page to verify the data flow (displays basic scalars).

4. Verification Plan

- **Automated:** Syntax check.
 - **Manual:**
 1. Run `docker compose up --build`.
 2. Visit `http://localhost:8000`.
 3. Upload 1, 2, or 3 Cabrillo files.
 4. Verify the "Dashboard" page loads and shows correct log metadata (Callsign, Score).
-

5. Surgical Changes

File: `web_app/config/settings.py`

Baseline: 0.102.0-Beta **Change:** Add `web_app.analyzer` to `INSTALLED_APPS`.

__CODE_BLOCK__python

... existing imports ...

```
INSTALLED_APPS = [ 'django.contrib.admin', 'django.contrib.auth',  
'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages',  
'django.contrib.staticfiles', 'web_app.analyzer', # New App ]
```

... existing middleware ...

CODE_BLOCK

File: `web_app/config/urls.py`

Baseline: 0.102.0-Beta **Change:** Include `analyzer.urls` for the root path.

__CODE_BLOCK__python from django.contrib import admin from
django.urls import path, include # Added include

```
urlpatterns = [ path('admin/', admin.site.urls), path('', include('web_app.analyzer.urls')),
# Root route ] CODE_BLOCK
```

File: web_app/analyzer/apps.py (New)

```
__CODE_BLOCK__python from django.apps import AppConfig

class AnalyzerConfig(AppConfig): default __auto__ field = 'django.db.models.BigAutoField'
name = 'web_app.analyzer' CODE_BLOCK
```

File: web_app/analyzer/forms.py (New)

```
__CODE_BLOCK__python from django import forms

class UploadLogForm(forms.Form): # Identity Agnostic Labels log1 =
forms.FileField( label="Log 1", required=True, widget=forms.FileInput(attrs={'class':
'form-control', 'accept': '.log,.cbr,.txt'}) ) log2 = forms.FileField( label="Log
2", required=False, widget=forms.FileInput(attrs={'class': 'form-control',
'accept': '.log,.cbr,.txt'}) ) log3 = forms.FileField( label="Log 3", re-
quired=False, widget=forms.FileInput(attrs={'class': 'form-control', 'accept':
'.log,.cbr,.txt'}) ) CODE_BLOCK
```

File: web_app/analyzer/urls.py (New)

```
__CODE_BLOCK__python from django.urls import path from . import
views

urlpatterns = [ path('', views.home, name='home'), path('analyze/',
views.analyze_logs, name='analyze'), ] CODE_BLOCK
```

File: web_app/analyzer/views.py (New)

```
__CODE_BLOCK__python import os import tempfile import shutil from
django.shortcuts import render, redirect from .forms import UploadLogForm
```

Import Core Logic

```
from contest_tools.log_manager import LogManager from contest_tools.data_aggregators.time_series
import TimeSeriesAggregator

def home(request): form = UploadLogForm() return render(request, 'ana-
lyzer/home.html', {'form': form})

def analyze_logs(request): if request.method == 'POST': form = UploadLog-
Form(request.POST, request.FILES) if form.is_valid(): # 1. Create Ephemeral
Context temp_dir = tempfile.mkdtemp() try: # 2. Save Uploads log_paths
= [] # Ensure data directory exists for CTY lookups if not mapped (Docker
handles this though)
```

```

files = [request.FILES.get('log1'), request.FILES.get('log2'), request.FILES.get('log3')]
files = [f for f in files if f] # Filter None

for f in files:
    file_path = os.path.join(temp_dir, f.name)
    with open(file_path, 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
    log_paths.append(file_path)

# 3. Process with LogManager
# Note: We rely on docker-compose env vars for CONTEST_INPUT_DIR to find CTY file
root_input = os.environ.get('CONTEST_INPUT_DIR', '/app/CONTEST_LOGS_REPORTS')

lm = LogManager()
# Load logs (auto-detect contest type)
lm.load_log_batch(log_paths, root_input, 'after')

# 4. Aggregate Data (Proof of Life)
ts_agg = TimeSeriesAggregator(lm.logs)
ts_data = ts_agg.get_time_series_data()

# Extract basic scalars for dashboard
context = {
    'logs': [],
    'contest_name': lm.logs[0].metadata.get('ContestName', 'Unknown'),
}

for call, data in ts_data['logs'].items():
    context['logs'].append({
        'callsign': call,
        'score': data['scalars']['points_sum'], # Using raw points for simple version
        'qsos': data['scalars']['net_qsos']
    })

return render(request, 'analyzer/dashboard.html', context)

except Exception as e:
    return render(request, 'analyzer/home.html', {'form': form, 'error': str(e)})
finally:
    # 5. Cleanup
    shutil.rmtree(temp_dir)

return redirect('home')
CODE_BLOCK

```

File: web_app/analyzer/templates/analyzer/base.html (New)

__CODE_BLOCK__html

```
<div class="container">
    {% block content %}{% endblock %}
</div>
```

```
<div class="container footer pb-4">
    &copy; 2025 Mark Bailey, KD4D | Powered by Django & Plotly
</div>
```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
```

File: web_app/analyzer/templates/analyzer/home.html (New)

__CODE_BLOCK__html {% extends "analyzer/base.html" %}

```
{% block content %}

    <form method="post" action="{% url 'analyze' %}" enctype="multipart/form-data">
        {% csrf_token %}

        <div class="row mb-3">
            <div class="col-md-6 mb-3 mb-md-0">
                <label class="form-label fw-bold">{{ form.log1.label }} <span class="text-danger">{{ form.log1.errors }}
```

</div>

File: `web_app/analyzer/templates/analyzer/dashboard.html` (New)

```
__CODE_BLOCK__html {% extends "analyzer/base.html" %}
{% block content %}
```

File: `web_app/analyzer/__init__.py` (New)

(Empty file)

6. Pre-Flight Check

- **Inputs:** `web_app/config/settings.py`, `web_app/config/urls.py` (Version 0.102.0-Beta).
- **Expected Outcome:** A functional Django app `web_app.analyzer` serving the upload form at the root URL.
- **Architecture Compliance:**
 - **Statelessness:** Confirmed. `views.py` uses `tempfile` and `shutil.rmtree` to ensure no persistence of uploaded logs.
 - **Identity Agnostic:** Confirmed. `forms.py` uses "Log 1", "Log 2", "Log 3" labeling.
- **Dependency Verification:** `LogManager` and `TimeSeriesAggregator` imports in `views.py` are valid relative to the project root and `PYTHONPATH` set in `Dockerfile`.
- **Surgical Modification Adherence:** `settings.py` and `urls.py` modifications are append-only or specific list insertions, preserving existing config.