ExoHabitAI: Milestone3-Documentation

Refer for more detailed Documentation – Link

1. Executive Summary

ExoHabitAI is a full-stack machine learning application designed to identify potentially habitable exoplanets from NASA's archive. It bridges the gap between complex astrophysical data and intuitive discovery by combining a **Hybrid Stacking Ensemble Model** (Backend) with a **Cinematic 3D Dashboard** (Frontend).

The system prioritizes **Scientific Recall**, using an optimized threshold of **0.0763** to ensure that no potential Earth-like worlds are overlooked.

2. System Architecture

2.1 High-Level Stack

- **Data Core:** Python, Pandas, Scikit-Learn (Milestones 1 & 2)
- **Backend API:** Flask, Joblib (Module 5)
- **Frontend Client:** React 18, Vite, Tailwind CSS (Module 6)
- **Communication:** RESTful API (JSON) over HTTP

2.2 Data Flow

1. **User Input:** Astronomer enters 39 planetary parameters via the React Dashboard.
2. **Transport:** Frontend sends a JSON payload to the Flask API (`POST /predict`).
3. **Validation:** Backend middleware (`utils.py`) verifies features and types.
4. **Inference:** The Stacking Model (XGBoost + Random Forest) calculates a probability.
5. **Decision:** If probability >= 0.0763, the planet is flagged as **Habitable**.
6. **Visualization:** React UI renders a "Confidence Gauge" and status badge.

---

3. Backend Module (Flask API)

3.1 Technical Specifications

- **Server:** Flask (Python 3.10+)
- **Model Loading:** `joblib` loads the `.pkl` file once at startup (Singleton pattern).
- **Security:** `flask_cors` enables secure cross-origin requests from the frontend.

3.2 API Endpoints

Endpoint	Method	Purpose	Request Body / Query
/status	GET	Health check & model status	N/A
/predict	POST	Single planet classification	JSON (39 features)
/predict/batch	POST	Bulk catalog analysis	{"planets": [...]}
/rank	GET	Retrieve top candidates	?top=10&min_score=0.9
/model/info	GET	Get model metadata	N/A

3.3 Input Validation Strategy

To prevent server crashes, the backend implements a "Bouncer" in utils.py:

- **Missing Features:** Returns 400 Bad Request with a list of missing keys.
- **Type Errors:** Returns 400 if non-numeric data is detected.
- **Empty Payload:** Returns 400 with a user-friendly error message.

---

4. Frontend Module (React Client)

4.1 Technical Specifications

- **Framework:** React 18 + Vite 5 (for lightning-fast builds)
- **Styling:** Tailwind CSS (Deep Space Theme: #0b0c10 background), Framer Motion
- **Visualization:** Chart.js (Probability distribution) & Lucide Icons

4.2 Key Features

- **Cinematic Home Page:** Features a 3D "Black Hole" hero section using Three.js and Framer Motion to engage users.
- **Smart Forms:** The `/predict` page groups inputs by "Scientific Drivers" (Mass, Radius, Flux) vs. "Advanced Parameters" (hidden by default).
- **Batch Processor:** A dedicated `/batch` page allows users to paste JSON catalogs and view a summary table of results.
- **Ranking Leaderboard:** Displays the "Top 10 Most Habitable Worlds" fetched live from the backend.

4.3 Integration Logic (`api.js`)

All API calls are centralized in a service layer to separate logic from UI components.

```
JavaScript

// src/services/api.js
import axios from 'axios';

const api = axios.create({
    baseURL: import.meta.env.VITE_API_URL || 'http://localhost:5000',
    headers: { 'Content-Type': 'application/json' }
});

export const apiService = {
    getStatus: () => api.get('/status'),
    predict: (data) => api.post('/predict', data),
    // ... other endpoints
};
```

---

5. Setup & Installation Guide

Step 1: Backend Setup

```
Bash

cd backend
python -m venv .venv
# Activate virtual environment
# Windows: .venv\Scripts\activate
# Mac/Linux: source .venv/bin/activate
pip install -r requirements.txt
python app.py
```

Verify: Visit `http://localhost:5000/status` to confirm the system is online.

Step 2: Frontend Setup

```
Bash

cd frontend
npm install
cp .env.example .env  # Ensure VITE_API_URL=http://localhost:5000
npm run dev
```

Access: Open `http://localhost:3000` to view the dashboard.

---

6. Testing & Validation

- **Unit Testing:** `pytest` suite covers all backend endpoints (`test_api.py`).
- **Integration Testing:** Confirmed that the React frontend correctly handles:
    - **Success:** Displays "Habitable" badge for Earth-like inputs.
    - **Failure:** Shows red error alerts if the backend is offline.
    - **Latency:** Average response time is < 200ms per request.

---

7. Conclusion

ExoHabitAI successfully delivers a robust, full-stack scientific tool. The backend ensures rigorous physical validation and high-recall discovery, while the frontend provides an accessible, professional interface for modern astronomers.