Exoplanet Habitability Prediction System

Backend Documentation

---

1. Introduction

The Exoplanet Habitability Prediction Backend is a Flask-based REST API designed to evaluate the habitability potential of exoplanets using a trained Machine Learning model (Random Forest). The system processes planetary and stellar parameters provided by the frontend, computes a habitability probability score, and returns a classification label.

This backend also provides a ranking service that lists the top habitable exoplanets based on pre-computed model scores.

---

2. Technology Stack

Component	Technology Used
Backend Framework	Flask
Machine Learning Model	Random Forest Classifier
Model Serialization	Joblib (.pkl)
Data Processing	Pandas
API Communication	REST (JSON)
CORS Handling	Flask-CORS

---

3. System Architecture

Frontend → Flask API → Utils (ML Logic) → Random Forest Model → Response (JSON)

---

4. API Endpoints

4.1 Home Endpoint

URL: /

Method: GET

Description:
Checks whether the API server is running successfully.

Response Format

{

"status": "success",

"message": "Exoplanet Habitability API is running"

}

4.2 Prediction Endpoint

URL: /predict

Method: POST

Description:
Predicts whether an exoplanet is habitable based on planetary and stellar input parameters.

---

5. Request Format

The API expects input data in **JSON format**.

Sample Request Body

{

"pl_bmasse": 1.2,

"pl_orbper": 365,

"pl_orbsmax": 1.01,

"pl_eqt": 280,

"pl_dens": 5.5,

"st_lum": 1.0,

"st_met": 0.02,

"star_F": 0,

"star_G": 1,

"star_K": 0

}

---

6. Feature Description

Feature Name	Description	Unit
pl_bmasse	Planet mass	Earth Mass
pl_orbper	Orbital period	Days
pl_orbsmax	Distance from star	AU
pl_eqt	Equilibrium temperature	Kelvin
pl_dens	Planet density	g/cm³
st_lum	Stellar luminosity	Solar Luminosity
st_met	Stellar metallicity	[Fe/H]
star_F	Star type F (one-hot)	Binary
star_G	Star type G (one-hot)	Binary
star_K	Star type K (one-hot)	Binary

---

7. Response Format

Success Response


```
{
  "status": "success",
  "prediction": {
    "habitability": "Habitable",
    "score": 0.6734
  }
}
```


Response Fields

Field	Description
habitability	Final classification label
score	Probability score (0–1)

Error Response

```
{
  "status": "error",
  "message": "No JSON data received"
}
```

---

8. Ranking Endpoint

URL: /rank

Method: GET

Description:
Returns the top 10 most habitable exoplanets from the ranked dataset.

Sample Response

```
{
  "status": "success",
  "top_habitable_exoplanets": [
    {
      "planet_name": "Kepler-442b",
      "habitability_score": 0.91
    }
  ]
}
```

---

9. Backend Theory

9.1 Machine Learning Model

A Random Forest Classifier is used for prediction. It is an ensemble learning technique that combines multiple decision trees to improve accuracy and reduce overfitting.

Advantages

- Handles nonlinear relationships
- Works well with tabular data
- Reduces variance via averaging
- Provides probability scores

9.2 Habitability Threshold

The model outputs a probability value between 0 and 1.

A custom threshold of **0.4** is used:

- Probability ≥ 0.4 → Habitable
- Probability < 0.4 → Not Habitable

This threshold was selected based on model evaluation to balance precision and recall.

9.3 Input Preprocessing

Before prediction:

1. Input JSON is converted to a Pandas DataFrame.
2. Missing features are filled with 0.
3. Feature order is aligned with training data.

This ensures compatibility with the trained model.

9.4 Model Serialization

The trained model is stored as:

random_forest.pkl

Joblib is used for fast loading during backend startup.

---

10. Error Handling

The backend includes exception handling for:

- Missing JSON input
- Model errors
- File not found errors
- CSV loading failures

All errors return structured JSON responses.

---

11. Testing the API

Using Postman / Thunder Client

Prediction Test

POST → /predict

Ranking Test

GET → /rank

---

12. Conclusion

The backend system efficiently integrates Machine Learning with a Flask API to deliver real-time exoplanet habitability predictions. Its modular design enables scalability, easy frontend integration, and future enhancements such as additional ML models or live NASA dataset connections.

Frontend UI Development

ExoHabitAI – Exoplanet Habitability Prediction System

---

1. Objective

The objective of the Frontend UI Development module is to design and implement a user-friendly, visually engaging, and responsive web interface that allows users to:

- Enter planetary and stellar parameters
- Submit data for habitability prediction
- View prediction results and habitability scores
- Interact with an immersive space-themed interface

The frontend communicates with the Flask backend API to fetch real-time machine learning predictions.

---

2. Frontend Folder Structure

The frontend follows a modular folder structure:

frontend/

│

├── index.html → UI structure & layout

├── style.css → Styling & responsiveness

└── script.js → Logic & API integration

Each file has a distinct responsibility to maintain separation of concerns.

---

3. UI Layout Design

The UI is designed using:

- HTML5
- Bootstrap 5
- Custom CSS
- Google Fonts

- Bootstrap Icons

The layout includes:

1. Loading screen
2. 3D animated space background
3. Header section
4. Input form panels
5. Prediction result panel
6. Fixed footer action buttons

Bootstrap Grid System is used to ensure responsive alignment of form elements.

---

4. Input Form Creation

The form is divided into two main sections:

4.1 Planetary Parameters

Input fields include:

- Planet Radius
- Radius Unit (Earth radii / km)
- Planet Mass
- Planet Density
- Orbital Period
- Orbital Distance
- Equilibrium Temperature

4.2 Stellar Parameters

Input fields include:

- Star Effective Temperature
- Star Luminosity
- Star Metallicity
- Star Type (F, G, K)

All inputs use:

- Labels for clarity
- Placeholders for guidance
- Required validation
- Numeric input restrictions

---

5. Client-Side Validation

Validation is implemented in JavaScript before sending data to the backend.

Validation checks include:

- Empty field detection
- Numeric parsing
- Positive value enforcement
- Star type selection requirement

If invalid input is detected, a user-friendly error alert is displayed.

---

6. Data Preprocessing

Before sending data to the backend:

Radius Conversion

If radius is entered in kilometers, it is converted to Earth radii:

Earth Radius = 6371 km

Star Type Encoding

Categorical star types are converted to one-hot encoding:

- star_F
- star_G
- star_K

This ensures compatibility with the machine learning model.

---

7. Backend Integration

Frontend communicates with Flask API using the Fetch API.

Endpoint Used

POST → /predict

Request Format

JSON payload containing planetary and stellar features.

Example:

```
{
  pl_bmasse: 1.1,
  pl_orbper: 365,
  pl_eqt: 280,
  st_lum: 1.0,
  star_G: 1
}
```

The request is sent asynchronously without reloading the page.

---

8. Prediction Result Display

Results are displayed dynamically using a dedicated result panel.

Displayed outputs:

- Habitability Status (Habitable / Not Habitable)
- Habitability Score (%)

Visualization components include:

- Status badge
- Score percentage text
- Animated progress bar
- Color-coded indicators

Animations are handled using GSAP for smooth transitions.

---

9. Error Handling

User-friendly error messages are shown for:

- Invalid inputs
- Missing values
- API failures
- Server errors

The error alert panel dynamically appears without refreshing the page.

---

10. 3D Background & Visual Experience

A cinematic space environment is implemented using Three.js.

Key elements include:

- Rotating Earth-like planet
- Atmospheric glow shader
- Cloud layer
- Starfield particles
- Cosmic dust effects
- Dynamic lighting system

This enhances user engagement and visual appeal.

---

11. Loader & Intro Animations

A mission-style loading screen is displayed during initialization.

Features:

- Planet loader animation
- Progress bar
- GSAP fade transitions

Intro animations animate:

- Header appearance
- Form panels

---

12. Styling & Responsiveness

Styling is implemented using:

- CSS variables (theme system)
- Glassmorphism panels
- Neon glow effects
- Gradient buttons
- Custom scrollbars

Responsive design ensures usability across:

- Mobile devices
- Tablets
- Desktops

Bootstrap grid and media queries handle layout adjustments.

---

13. Integration Testing

Complete system testing flow:

User Input → Frontend Form → API Request → Flask Backend → ML Model → Response → UI Display

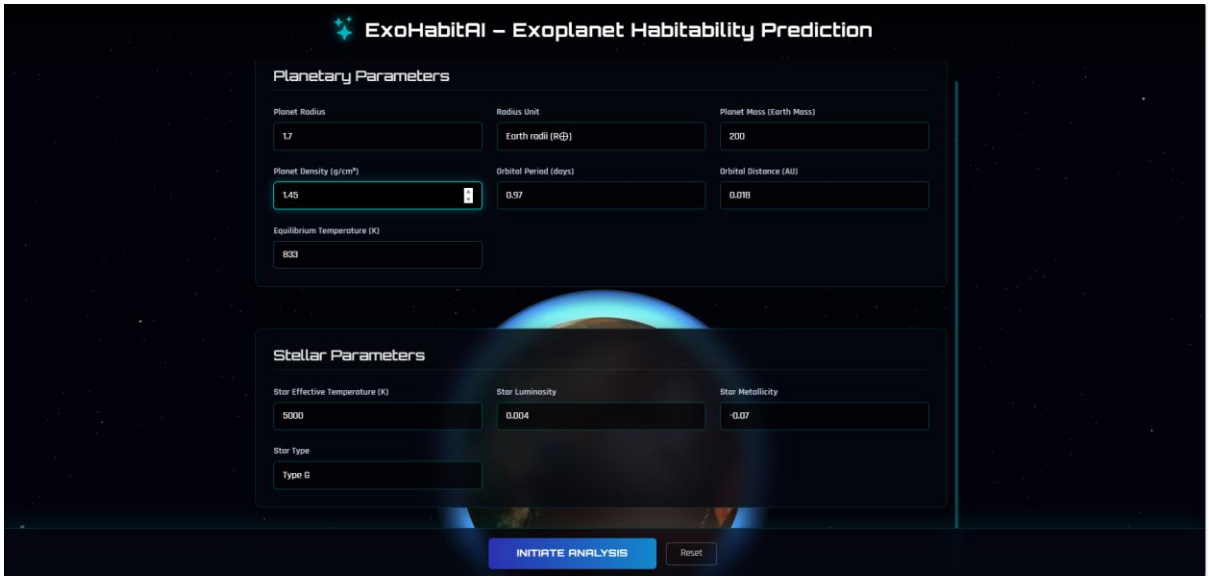
Testing verifies:

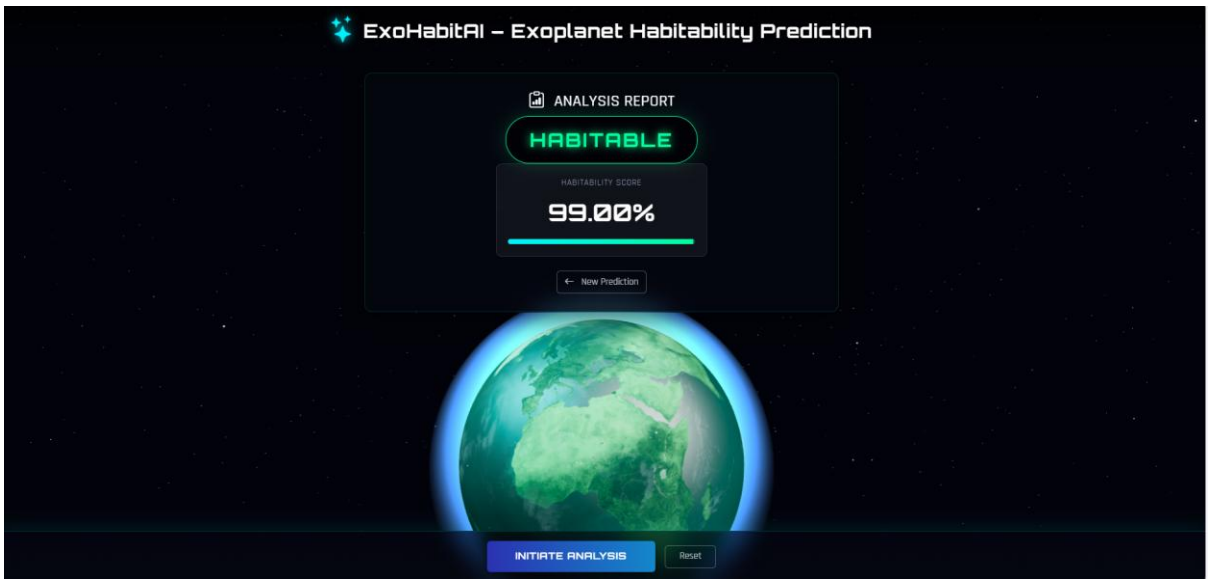
- Data transmission accuracy
- Prediction correctness
- UI rendering
- Error handling
- Performance

---

14. Screenshots Section

The following UI screenshots are included to demonstrate the working frontend interface:





14.1 Input Form Interface

This screen displays the planetary and stellar parameter input sections where users enter exoplanet data before prediction.

Key visible components:

- Planetary Parameters panel
- Stellar Parameters panel
- Numeric input fields

- Dropdown star type selector
- Predict Habitability button
- Reset button
- 3D planet background visualization

14.2 Prediction Result Interface

This screen shows the generated habitability analysis after submitting input data.

Displayed elements include:

- Analysis Report header
- Habitability status label (Habitable / Not Habitable)
- Habitability score percentage
- Animated progress bar
- New Prediction button
- Dynamic planet visualization

These screenshots validate successful frontend–backend integration and result rendering.

---

15. Conclusion

The frontend system successfully delivers an interactive and visually immersive platform for exoplanet habitability analysis. By integrating modern UI/UX design, 3D visualization, and real-time backend communication, the system provides both scientific insight and engaging user experience.
