BACKEND MODULE

1. Overview

The backend of **ExoHabitAI** is built to make a trained machine-learning model usable in a real application. Instead of running predictions only inside a notebook, the backend exposes the model through a **Flask REST API**. This allows any client (browser, Postman, or frontend application) to send exoplanet data and receive a habitability prediction in a clean JSON format.

---

2. Backend Structure and Design

- **app.py**
  - This is the main Flask application.
  - It defines API routes such as / and /predict.
  - It loads the trained Random Forest model and performs predictions.
- **utils.py**
  - This file contains helper functions.
  - Its main responsibility is input validation.
  - Keeping validation logic here keeps app.py clean and readable.
- **models/random_forest.pkl**
  - This file stores the trained Random Forest model.
  - It is loaded using joblib when the backend starts.

---

3. Model Loading and Integration

The trained Random Forest model is saved as a .pkl file during the training phase. In the backend, this model is loaded only once at startup:

```
model = joblib.load("../models/random_forest.pkl")
```

Loading the model at startup improves performance because the model does not need to be reloaded for every request.

---

4. Input Features

The model was trained using **21 planetary and stellar features**. These features describe physical, orbital, and stellar characteristics of an exoplanet.

During prediction: - All 21 features must be provided - Feature names must exactly match the training features - Feature order must remain the same as during training

This ensures consistency between training and prediction.

5. API Endpoints

5.1 Home Endpoint (`/`)

- **Method:** GET
- **Purpose:** Check whether the backend server is running

Example response:

```
{
  "message": "ExoHabitAI Backend is running successfully"
}
```

This endpoint is mainly used for quick health checks.

5.2 Prediction Endpoint (`/predict`)

- **Method:** POST
- **Purpose:** Predict the habitability of an exoplanet

The client sends all 21 features in JSON format. The backend: 1. Validates the input 2. Converts the input into a numerical feature vector 3. Passes the data to the trained model 4. Returns the prediction and probability score

Example response:

```
{
  "status": "success",
  "prediction": 1,
  "habitability_score": 0.589
}
```

Here: - `prediction = 1` indicates a potentially habitable planet - `habitability_score` represents the confidence of the model

6. Input Validation and Error Handling

Before making predictions, the backend checks: - Whether all required features are present - Whether the values are numeric

If any feature is missing or invalid, the backend returns a clear error message:

```
{
  "status": "error",
  "message": "Missing field: pl_rade"
}
```

This prevents incorrect or misleading predictions and makes the API reliable.

---

7. Testing the Backend

The backend was tested using: - **Browser** for GET requests - **PowerShell or Postman** for POST requests

Testing confirmed: - Successful model loading - Correct handling of valid inputs - Proper error messages for invalid inputs

---

8. Conclusion

The ExoHabitAI backend successfully bridges the gap between machine-learning experimentation and real-world application. By using Flask, structured APIs, and strict validation, the backend provides a reliable and scalable way to predict exoplanet habitability.

FRONTEND MODULE

1. Purpose of Frontend

The frontend represents the **interaction and visualization layer** of ExoHabitAI. Its goal is not only to collect user input but also to **engage the user visually** and present AI results in an intuitive way.

Unlike basic forms, this frontend uses a **live animated galaxy environment** to reflect the project's space-science theme.

---

2. Frontend Technologies Used

Technology	Purpose
HTML5	UI structure
CSS3	Glassmorphism styling
JavaScript	Logic and API communication
Three.js	Real-time galaxy animation
WebGL	GPU-accelerated rendering

---

3. Frontend Structure

- index.html defines layout and canvas
- style.css defines visual design
- script.js controls animation and backend calls

---

4. Galaxy Animation

Instead of a static background, a **real-time animated galaxy** was implemented using **Three.js**. Thousands of particles represent stars distributed in 3D space.

Key concepts applied:

- Buffer geometry for performance
- Continuous rotation to simulate galaxy motion
- Responsive resizing based on screen size

This makes the frontend feel **live and scientific**, rather than decorative.

---

5. Frontend - Backend Communication

The frontend uses the **Fetch API** to send user input to the backend:

```
fetch("http://127.0.0.1:5000/predict", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(data)

})
```

- Data is sent in JSON format
- Backend processes the request
- Response is returned asynchronously

This allows predictions without page reloads.

---

6. Result Visualization

Once the backend responds:

- Habitability status is displayed
- Confidence score is shown
- UI updates dynamically

This immediate feedback improves usability and clarity.

---

7. Integration of Backend and Frontend

The real strength of ExoHabitAI lies in its **integration**:

- Backend provides intelligence
- Frontend provides experience

Both modules were developed independently but designed to work together through well-defined APIs.

This modular approach ensures:

- Easy debugging
- Independent upgrades
- Clean separation of concerns

---

8. Conclusion

ExoHabitAI successfully demonstrates how machine learning, backend APIs, and advanced frontend visualization can be combined into a single intelligent system. The backend performs accurate habitability prediction, while the frontend delivers an immersive and engaging user experience.