MILE STONE -3 : BACKEND AND FRONTEND

BACKEND MODULE

The backend module of **ExoHabitAI** is designed to deploy a trained machine-learning model as a usable web service. Instead of limiting predictions to a Jupyter notebook environment, the backend exposes the model through a Flask-based REST API. This allows external clients such as web browsers, frontend applications, or API testing tools to send exoplanet data and receive habitability predictions in a structured JSON format.The backend acts as a bridge between the trained model and real-world applications.

Backend Architecture and File Structure

The backend follows a modular design to improve readability, maintainability, and scalability.

app.py

- Acts as the main Flask application.
- Defines all API routes such as / and /predict.
- Loads the trained Random Forest model at startup.
- Handles incoming requests, performs predictions, and returns results as JSON responses.

utils.py

- Contains helper and utility functions.
- Responsible for:
    - Input validation
    - Feature preprocessing
    - Feature engineering required by the trained model
- Separating this logic keeps app.py clean and focused on request handling.

Model File

- The trained model is stored as a serialized file:
- random_forest.pkl
- The model was trained during the machine-learning phase and saved using joblib.

Model Loading and Integration

The Random Forest model is loaded once at backend startup using the joblib library.

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

Loading the model at startup improves performance by avoiding repeated disk access and ensures faster response times for prediction requests.

Input Features

The machine-learning model was trained using multiple planetary and stellar features describing physical, orbital, and stellar properties of exoplanets.

During prediction:

- All required features must be provided
- Feature names must exactly match those used during training
- Feature order must remain consistent

Maintaining strict feature consistency ensures reliable and accurate predictions.

API Endpoints

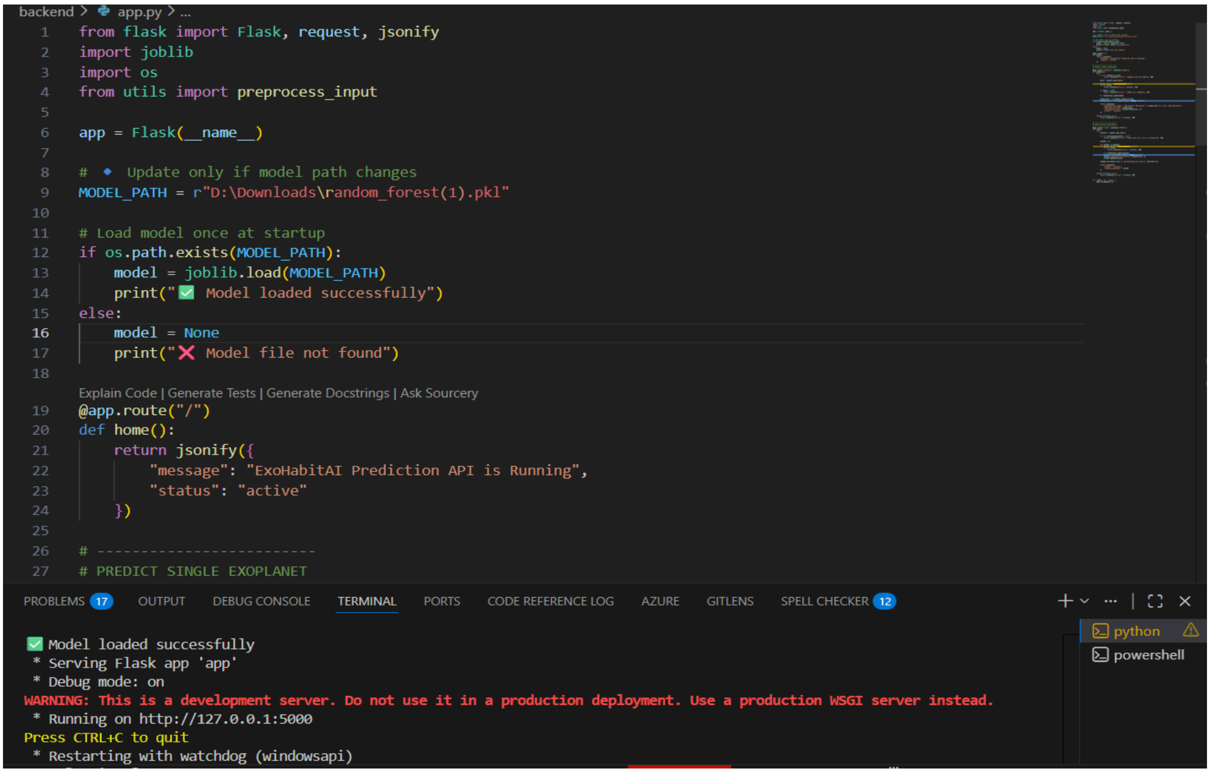
Home Endpoint (/)

- **Method:** GET
- **Purpose:** Verifies that the backend server is running

Example Response:

```
{
  "message": "ExoHabitAI Prediction API is Running",
  "status": "active"
}
```

This endpoint is mainly used for health checks and testing connectivity.



Prediction Endpoint (/predict)

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

Workflow:

1. Accepts input data in JSON format
2. Validates the presence and correctness of required features
3. Preprocesses and converts the input into a model-ready feature vector
4. Passes the data to the trained Random Forest model
5. Returns the prediction result and confidence score


```
{
  "status": "success",
  "habitability_label": "Potentially Habitable",
  "confidence_score": 0.59
}
```


Input Validation and Error Handling

To ensure reliable predictions, the backend performs strict input validation before inference.

Validation checks include:

- Verification that all required fields are present
- Ensuring numerical values are correctly formatted

If invalid input is detected, the backend returns a clear and informative error message.

Backend Testing

The backend was tested using multiple methods:

- Web browser for testing the home endpoint
- PowerShell for sending POST requests to the prediction endpoint

Testing confirmed:

- Successful model loading
- Correct predictions for valid input
- Proper error handling for missing or invalid data

Conclusion

The ExoHabitAI backend successfully transforms a trained machine-learning model into a real-world application. By combining Flask, structured REST APIs, modular design, and strict input validation, the backend provides a reliable, scalable, and efficient system for predicting exoplanet habitability.

FRONTEND MODULE

Purpose of the Frontend

The frontend of ExoHabitAI serves as the user interaction and visualization layer. Its primary goals are:

- Collecting planetary and stellar input data
- Communicating with the backend prediction API
- Presenting AI results in an engaging and intuitive manner

Frontend Structure

- **index.html**
  Defines layout, form inputs, and canvas elements.
- **style.css**
  Controls visual styling, themes, and layout aesthetics.
- **script.js**
  Handles animations, user interaction, and backend communication.

This structure ensures clarity and maintainability.

Galaxy Animation

The frontend features a real-time animated galaxy background implemented using Three.js.
Thousands of particles represent stars distributed in 3D space.

Key techniques applied:

- Buffer geometry for performance optimization
- Continuous rotation to simulate galactic motion
- Responsive resizing for different screen dimensions

Frontend–Backend Communication

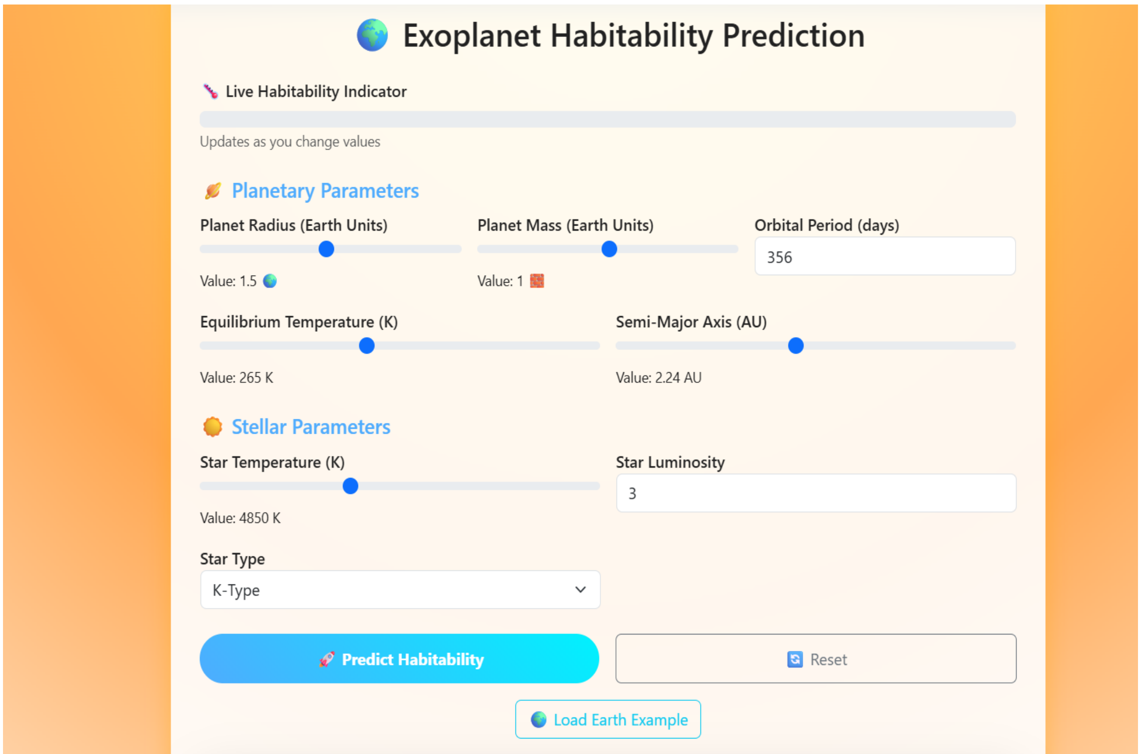
The frontend communicates with the backend using the Fetch API

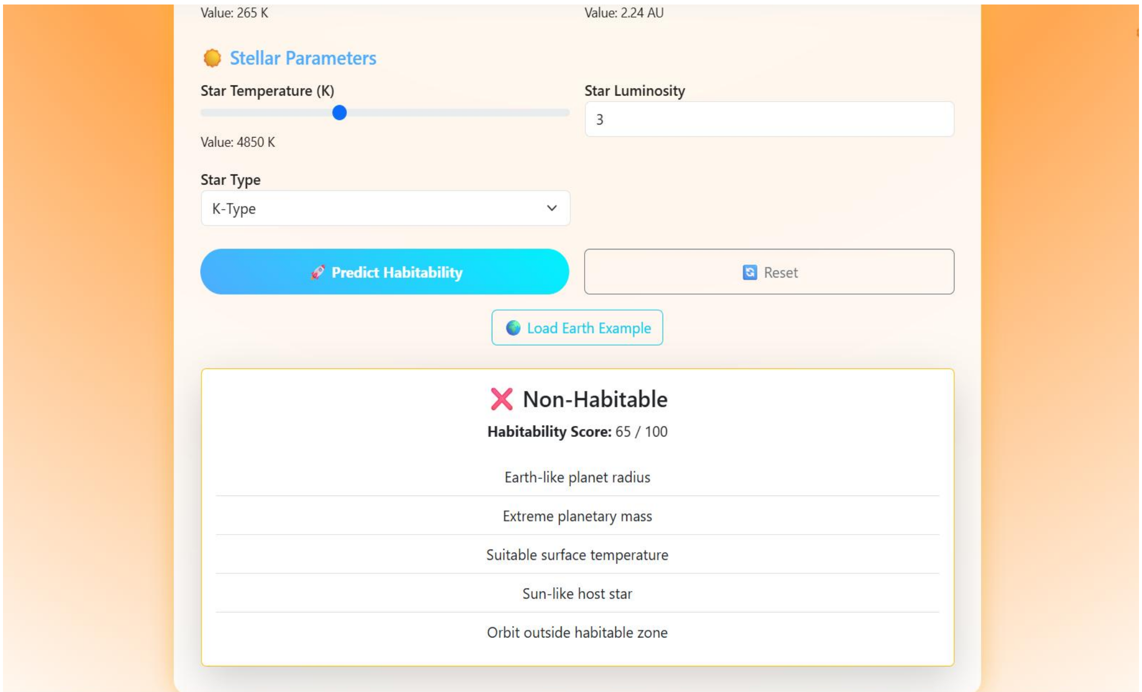
- Sends user input as JSON
- Uses asynchronous requests
- Receives predictions without page reloads
- This allows a smooth and responsive user experience.

Result Visualization

After receiving the backend response:

- Habitability status is displayed clearly
- Confidence score is shown numerically
- UI components update dynamically.





Integration of Backend and Frontend

The strength of ExoHabitAI lies in its modular integration:

- Backend provides intelligence through machine learning
- Frontend delivers visualization and interaction

Both modules operate independently but communicate through well-defined APIs, ensuring:

- Clean separation of concerns
- Easier debugging
- Future scalability

Frontend Conclusion

The ExoHabitAI frontend successfully combines advanced visualization with real-time AI interaction. By integrating Three.js animations with a responsive prediction interface, it delivers an immersive and educational user experience while maintaining technical robustness.