Module 5: Backend API Integration (Flask)

Objective

The objective of this module is to deploy the trained machine learning model as a backend service using **Flask**.
The backend exposes REST APIs that accept exoplanet parameters as input and return **habitability predictions and rankings**.

This allows the machine learning model to be accessed programmatically by external systems such as web or mobile applications.

Backend Environment Setup

The backend was developed using Python with the Flask framework.

Technologies Used

- Python 3.x
- Flask
- NumPy
- Pandas
- Joblib
- XGBoost (for model inference)

All required libraries were installed using `pip`, and the environment was verified before starting development.

Backend Folder Structure

The backend follows a clean and modular structure:

```
backend/
│
├── app.py                # Main Flask application
├── utils.py              # Input validation and preprocessing utilities
├── models/
│   └── habitability_model.pkl
├── feature_columns.pkl
```

This structure ensures separation of concerns and easy maintainability.

Model Loading

The trained machine learning model and feature configuration are loaded at the time the Flask server starts.

- The trained model is stored as a `.pkl` file.
- Feature column ordering is preserved using `feature_columns.pkl`.
- This guarantees consistent input format between training and inference.

The model is loaded once to improve performance and avoid repeated disk access.

Flask Application Setup

A Flask application is initialized in `app.py`.

- JSON request and response handling is enabled.
- Routes are defined using Flask decorators.
- The application runs on `localhost` during development.

API Endpoints Design

1. `/predict` Endpoint

Method: `POST`

Description:
Predicts whether an exoplanet is habitable based on given parameters.

Input Format (JSON):

```
{
  "pl_rade": 1.1,
  "pl_bmasse": 0.9,
  "pl_orbper": 365,
  "pl_eqt": 288,
  "st_spectype": "G"
```

```
}
```

Output Format (JSON):

```
{
  "prediction": 0
}
```

- `prediction = 1` → Habitable
- `prediction = 0` → Not Habitable

2. `/rank` Endpoint

Method: `POST`

Description:
Ranks multiple exoplanets based on their habitability score.

Input Format (JSON):

```
{
  "planets": [
    {
      "name": "EarthLike",
      "pl_rade": 1.1,
      "pl_bmasse": 1.0,
      "pl_orbper": 365,
      "pl_eqt": 288,
      "st_spectype": "G"
```

```
    },
    {
      "name": "HotPlanet",
      "pl_rade": 3.5,
      "pl_bmasse": 8.0,
      "pl_orbper": 50,
      "pl_eqt": 900,
      "st_spectype": "M"
    }
  ]
}
```


Output Format (JSON):


```
{
  "status": "success",
  "ranked_planets": [
    {
      "name": "EarthLike",
      "score": 0.0
    },
    {
      "name": "HotPlanet",
      "score": 0.0
    }
  ]
}
```


Planets are sorted in descending order of habitability score.

Input Validation

Input validation is handled using utility functions in `utils.py`.

- Ensures required parameters are present
- Validates data types
- Prevents invalid or malformed requests
- Returns meaningful error messages when validation fails

This improves API reliability and robustness.

Response Structure

All API responses are returned in **JSON format** and include:

- Prediction or ranking result
- Status indicator
- Structured and readable output

This makes the backend easy to integrate with frontend systems.

Backend Testing

The backend APIs were tested using:

- Browser (for basic route testing)
- PowerShell (`Invoke-RestMethod`)
- Manual JSON requests

All endpoints were verified to work correctly without crashes.

Conclusion

In this module, a complete backend system was successfully developed using Flask.
The trained machine learning model was integrated and exposed via REST APIs for prediction and ranking of exoplanets.

This backend serves as a bridge between the machine learning model and real-world applications, making the system scalable and deployable.