AstroMood Deployment Plan Document

Sümeyye Sıla Altay 231101077
Esmanur Ulu 231101024
Zeynep Yetkin 231101042
Hazal Epözdemir 231101037

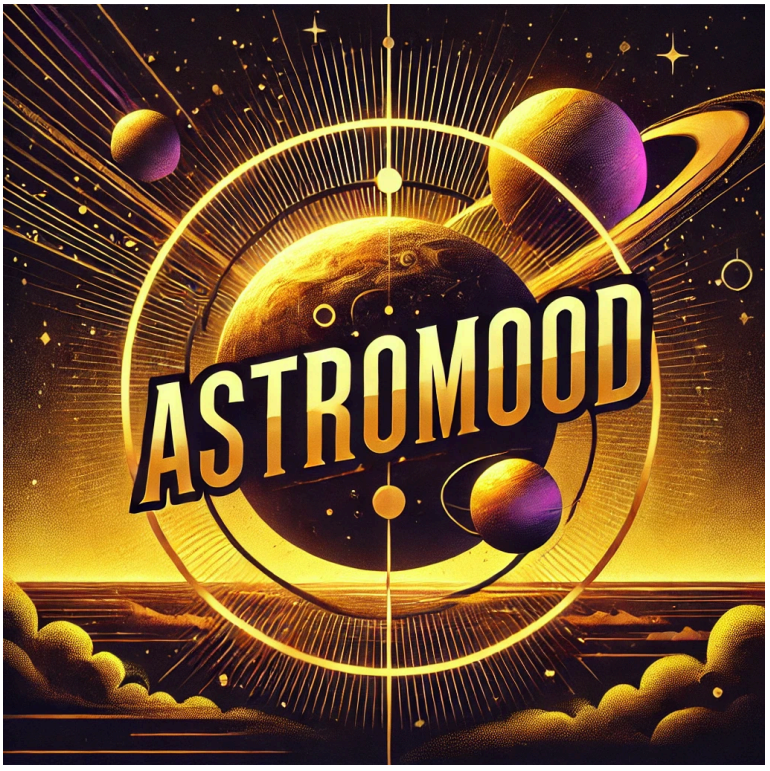

Table of Contents

Document Specific Task Matrix

Task	Team Member Responsible
Deployment Overview	Hazal Epözdemir, Sümeyye Sıla Altay
Deployment Process	Zeynep Yetkin, Esmanur Ulu
Configuration Plan	Hazal Epözdemir, Sümeyye Sıla Altay, Zeynep Yetkin, Esmanur Ulu

---

1. Introduction

This document outlines the deployment plan for the AstroMood project's demo presentation. The project consists of a **Flask-based backend** and a **JavaScript-based frontend**, ensuring a smooth integration between the logic and the user interface. The system is managed through version control with **GitHub** and follows continuous integration practices to ensure stability and reliability.

2. Deployment Overview

The project will be deployed using Azure Web Apps for the Flask backend, combined with GitHub Actions for automated deployment. The following tools and services are involved in the deployment process:

- **Azure Web App**: A fully managed platform for hosting web applications, providing automatic scaling, and handling infrastructure management.
- **GitHub Actions**: A CI/CD tool used to automate the build, test, and deployment process from the GitHub repository to Azure.
- **Version Control**: GitHub for managing the project's source code and changes.
- **Backend**: Flask (Python)
- **Frontend**: JavaScript, HTML, CSS (deployed via Vercel)
- **CI/CD Tools**: GitHub Actions for building, testing, and deploying the app automatically.

Deployment Environment

- **Backend**: Flask (Python)
- **Frontend**: JavaScript, HTML, CSS
- **Version Control**: GitHub
- **Deployment Platform**: **Azure Web App** (for backend Flask application)
- **CI/CD Tools**: GitHub Actions for CI/CD automation

---

3. Deployment Process

The deployment process consists of the following steps:

Code Preparation and Commit

- Ensure all code updates pass unit and integration tests before being committed to the GitHub repository.
- Each team member's contributions are tracked using descriptive commit messages.

Continuous Integration and Build

- Upon each commit to the repository, GitHub Actions automatically triggers the build and test process.
- The Flask backend is built, dependencies are installed, and tests are run via GitHub Actions.
- If the tests pass, the build is marked as successful, and the deployment pipeline is triggered.

Deployment Script and Deployment

- The deployment script in the GitHub repository automatically triggers the deployment process to Azure Web App once the build process completes successfully.
- Azure Web App will host the Flask backend.
- Configuration files such as environment variables (for API keys, database credentials, etc.) will be loaded automatically during deployment through Azure's environment variable management.

Post-Deployment Verification

- After deployment, both automated smoke tests and manual verification will be performed to ensure that the application is working as expected.
- In case of any issues, the team will address them promptly and initiate a new deployment.

---

4. Configuration Plan

Backend (Flask)

- **Dependencies**: The required Python packages (e.g., Flask, requests, etc.) will be listed in a `requirements.txt` file. GitHub Actions will install these dependencies during the CI/CD pipeline execution.
- **Environment Variables**: Sensitive data (e.g., API keys, database credentials) and environment-specific settings (debug mode, port numbers) will be managed using Azure's environment variable system.
    - The `.env` file should be used locally but will not be included in the repository. Instead, Azure's environment variables will handle this in production.
- **Flask Configuration**: The application will be configured to work with Azure Web Apps, ensuring it listens on the correct port and integrates with the Azure infrastructure.

Frontend (JavaScript, HTML, CSS)

- **Static Files**: The static files (JavaScript, CSS, images) are handled by the Flask backend, and the frontend will be served alongside the backend.
- **Flask Templates**: The Flask backend will render HTML templates dynamically based on the data sent from the backend to the frontend.

CI/CD Pipeline with GitHub Actions

- **Workflow**: The GitHub Actions pipeline will be set up to automate the following steps:
    1. Install dependencies (`requirements.txt` for Python packages).
    2. Run tests to ensure that the code is working as expected.
    3. Build and deploy the application to Azure Web Apps.
- **Security Configurations**: Ensure that the necessary security measures such as SSL certificates and firewall settings are applied in Azure to protect the backend.
- **Environment Setup**: The GitHub Actions workflow will manage environment configurations and automatically deploy to the correct Azure environment (development, staging, or production).

---

5. Logs and Outputs

- **Logs**: Logs of the deployment process, including errors and warnings, will be captured in Azure's monitoring tools and GitHub Actions logs.
- **Outputs**: Outputs from automated tests, deployment success messages, and status will be stored for future reference.