High Level Design Document

Introduction

This High Level Design (HLD) document outlines the architecture and core components of **TitanicTree - Survival Prediction Web App**. The application enables users to upload Titanic passenger data, predicts survival using a Decision Tree model, visualizes the model, and displays evaluation metrics. The system is built for educational purposes and is deployable via Docker for ease of hosting.

1. System Architecture Overview

Architecture Description:
The system is a modular Flask web application. Users interact via a web UI, which communicates with backend services for data processing, model inference, visualization, and metrics display. The application is containerized using Docker for deployment.

Main System Modules and Roles:

Module	Role
Web UI	User interface for data upload, results, and visualization
Flask API Backend	Handles HTTP requests, orchestrates workflow
Data Processing	Cleans and preprocesses uploaded data
ML Model (Decision Tree)	Loads/trains model, performs predictions
Visualization	Generates model and metrics visualizations
Docker Container	Encapsulates app and dependencies for deployment

2. Component Interactions

Sequence Step	Interaction Description
1	User uploads data via Web UI
2	Flask API receives file, passes to Data Processing
3	Processed data sent to ML Model for prediction
4	Model returns predictions and evaluation metrics to Flask API
5	Visualization module generates model/metrics visuals
6	Flask API sends results and visuals to Web UI for display

3. Data Flow Overview

Data Source	Flow Direction	Destination	Purpose
User CSV Upload	Web UI → Flask API	Data Processing	Data ingestion and preprocessing
Processed Data	Data Processing → ML	ML Model	Prediction and evaluation
Predictions/Metrics	ML Model → Flask API	Visualization	Visualization generation
Visuals/Results	Flask API → Web UI	User	Display of predictions and insights

4. Technology Stack

Layer/Function	Technology/Framework
Web Framework	Flask (Python)
Machine Learning	scikit-learn (Decision Tree)
Data Processing	pandas, numpy
Visualization	matplotlib, scikit-learn
Containerization	Docker

5. Scalability & Reliability

- **Scalability:**
  The application is stateless and containerized, enabling horizontal scaling via Docker orchestration (e.g., Docker Compose, Kubernetes).
- **Reliability:**
  Input validation and error handling are implemented at the API and data processing layers. Docker ensures consistent deployment environments.
- **Security:**
  File uploads are sanitized and restricted to CSV format. No sensitive data is stored.

End of Document