✅ Architecture Design Document Template

Project Title: Cryptocurrency Liquidity Prediction Web App
Version: 1.0
Author: Gaurav Singh
Email: gauravsin333@gmail.com

Table of Contents

---

1. Abstract

This document outlines the architectural framework for the "Cryptocurrency Liquidity Prediction Web App." The system uses machine learning techniques to predict liquidity scores based on historical market data, price trends, volume, and engineered features. It aims to assist investors, traders, and analysts with real-time insights to optimize trading strategies.

2. System Overview

The web application is built using a modular microservice architecture, with a clear separation between the data pipeline, model training, model inference, and frontend interface. The system uses XGBoost for model training and Streamlit for deployment.

3. Architecture Diagram

(Will provide visual after confirming next step)
Key Layers:

- Data Collection Layer
- Preprocessing & Feature Engineering
- ML Training Pipeline
- Inference Engine
- Web Frontend (Streamlit)
- REST API (Flask for model serving, if decoupled)

4. Component Description

Data Ingestion: Pulls real-time and historical data via APIs or CSV uploads.
ETL Pipeline: Cleans, normalizes, and engineers liquidity-specific features.
Model Trainer: XGBoost-based ML pipeline, with support for hyperparameter tuning.
Model Server: REST API endpoint to serve predictions.
Frontend: Simple UI for inputting data and displaying predictions.
Database (Optional): Stores historical predictions, input records, and model metrics.

5. Technology Stack

- **Frontend**: Streamlit
- **Backend**: Flask
- **ML Framework**: XGBoost, Scikit-learn
- **Language**: Python 3.x
- **Hosting**: Streamlit Cloud / AWS / Heroku
- **Version Control**: GitHub
- **Others**: Pandas, NumPy, Matplotlib/Seaborn for EDA

6. Data Flow Diagram

1. User inputs data on the frontend.

2. Data sent to backend for preprocessing.
3. Preprocessed data passed to XGBoost model.
4. Prediction result returned and displayed on frontend.

7. Deployment Architecture

- Dockerized Flask backend (optional)
- Streamlit app hosted on Streamlit Cloud
- GitHub Actions for CI/CD (optional)
- API security via token-auth (optional)

8. Security Considerations

- Secure endpoints (if using APIs)
- No sensitive user data stored
- Input validation at both frontend and backend

9. Future Scope

- Add more advanced models like LSTM for temporal analysis
- Deploy with auto-scaling on AWS Lambda or GCP
- Add portfolio simulation tools for user engagement

10. Conclusion

This architectural design serves as a blueprint for developing a robust and scalable cryptocurrency liquidity prediction platform. It emphasizes modularity, simplicity, and rapid deployment while maintaining room for future improvements.