# Low-Level Design (LLD)

## Restaurant Rating Prediction

**Revision Number: 1.0**

**Last date of revision: 21/02/2025**



iNeuron

## Document Version Control:

| Date Issued | Version | Description | Author |
|---|---|---|---|
| 21/02/2025 | **1** | Initial LLD-V1.0 | Alieh Hassanzadeh |
| | | | |
| | | | |

**Reviews:**

| Version | Date | Reviewer | Comments |
|---|---|---|---|
| | | | |
| | | | |

## Contents

# 1. Introduction

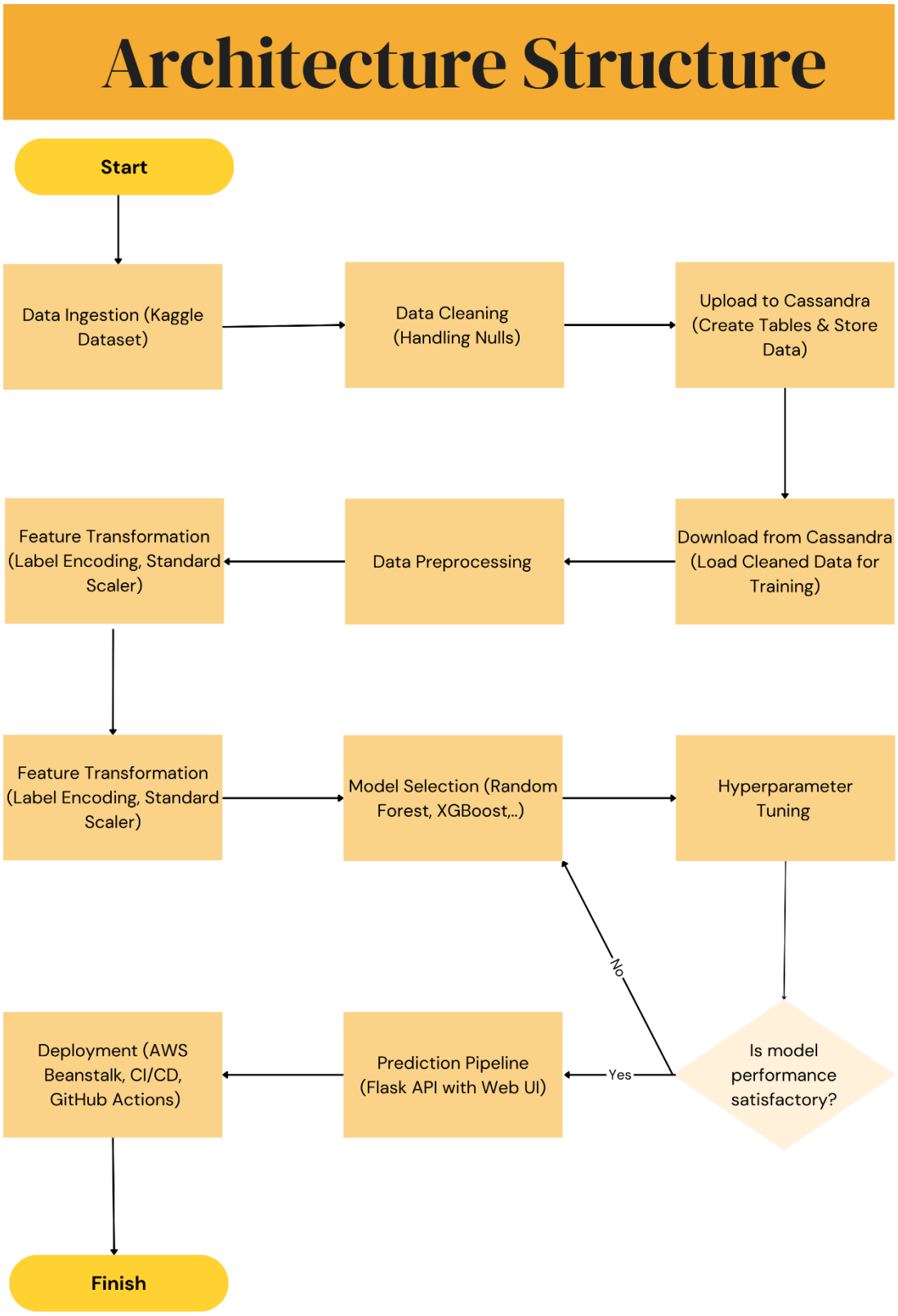
## 1.1. What is a Low-Level Design Document?

The goal of a Low-Level Design (LLD) document is to provide a detailed internal design of the system for Restaurant Rating Prediction. LLD includes class diagrams, method specifications, and relationships between different components of the system. It serves as a blueprint for developers to implement the system directly from the document.

## 1.2. Scope

Low-Level Design (LLD) is a component-level design process that refines the high-level architecture into detailed software components. It includes the design of data structures, system architecture, source code specifications, and performance optimization strategies. In this project, LLD defines the flow of data, model selection, training process, and deployment pipeline to ensure the system functions efficiently and accurately.

## 2. Architecture



# 3. Architecture Description

## 3.1. Data Ingestion

The dataset is sourced from Kaggle, containing structured information about restaurants, such as location, cuisine type, cost for two, votes, and online ordering availability. Since the dataset is static, no additional data collection or web scraping is required.

## 3.2. Data Cleaning

To ensure data consistency, missing values are handled, and any data inconsistencies are corrected.

## 3.3. Data Storage in Cassandra

- **Database Setup:** A Cassandra database is created and connected to the pipeline.
- **Table Creation:** A structured table is defined to store restaurant data efficiently.
- **Data Insertion:** The cleaned dataset is inserted into Cassandra for future retrieval.

## 3.4. Data Retrieval from Cassandra

The structured dataset is retrieved from Cassandra for further processing before model training.

## 3.5. Data Preprocessing

Before training the model, the following preprocessing steps are applied:

- **Handling Missing Values:** Missing values are either filled or removed.
- **Feature Selection:** Unnecessary features are dropped to improve model performance.

## 3.6. Feature Transformation

To enhance model accuracy, categorical attributes are encoded using label encoding, and numerical features are scaled using a standard scaler.

## 3.7. Model Training & Hyperparameter Tuning

- **Model Selection:** Machine learning models like Random Forest, XGBoost, and others are trained on the transformed dataset.
- **Hyperparameter Tuning:** GridSearchCV is applied to optimize model performance.

## 3.8. Model Evaluation

The trained model is evaluated using metrics such $R^2$ score to determine its accuracy and performance.

## 3.9. Prediction Pipeline & Deployment

- **Flask API:** A Flask-based API is built to handle user input and process predictions.
- **Web UI:** A simple web interface is designed to display the predicted restaurant ratings.
- **Deployment:** The final model and API are deployed on AWS Beanstalk using CI/CD pipelines with GitHub Actions and CodePipeline for automation.

# 4. Unit Test Cases

| Test Case Description | Pre-Requisite | Expected Result |
|---|---|---|
| **Verify whether the Application URL is accessible to the user** | 1. Application URL should be defined | Application URL should be accessible to the user |
| **Verify whether the Application loads completely for the user when the URL is accessed** | 1. Application URL is accessible<br>2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| **Verify whether user is able to successfully login to the application** | 1. The application is accessible via the browser.<br>2. The user has reached the welcome page.<br>3. The user clicks on the "Get Started with Your Prediction" button, which redirects them to the login form page. | The user should be redirected to the prediction form page, where they can successfully see all the input fields (e.g., for restaurant details, cost, etc.) and be able to fill them out. |
| **Verify whether the user is able to edit and fill out all input fields on the prediction form page.** | 1. The application is accessible via the browser.<br>2. The user has reached the prediction form page after clicking on the "Get Started with Your Prediction" button.<br>3. The input fields are visible and displayed correctly on the form. | The user should be able to successfully edit and fill out all the input fields on the prediction form page (e.g., name, location, cost, etc.). |

| | | |
|---|---|---|
| **Verify whether the user gets the Submit button to submit the inputs on the prediction form page.** | 1. The application is accessible via the browser.<br>2. The user has reached the prediction form page after clicking on the "Get Started with Your Prediction" button.<br>3. The input fields are visible and the user has filled them out. | The Submit button should be visible and enabled, allowing the user to submit the filled-out form. |
| **Verify whether the user is presented with recommended results after clicking the Submit button.** | 1. The application is accessible.<br>2. The user is on the prediction form page.<br>3. The input fields are visible and filled out.<br>4. The Submit button is enabled.<br>5. The user has clicked Submit. | After clicking Submit, the system should process the inputs and display the recommended restaurant rating results. |
| **Verify whether the recommended results are in accordance with the selections the user made.** | 1. The user has accessed the application and filled out the prediction form.<br>2. The user has submitted the form and received recommended results. | The recommended restaurant rating results should match the user's input selections. |