Low Level Design

## **Thyroid Disease Detection System**

| Written By | Shubham Chaudhary |
| --- | --- |
| Document Version | 1.1 |
| Last Revised | 12/12/2024 |

# Contents

Document Version control----------------------------------------------------------2

1 Introduction-------------------------------------------------------------------------4

1.1 What is Low-Level Design Document--------------------------------------4

1.2 Scope------------------------------------------------------------------------------4

2 Architecture--------------------------------------------------------------------------5

3 Architecture Description--------------------------------------------------------- 6

3.1 Data Description ---------------------------------------------------------------6

3.2 Export Data from DB to CSV for training----------------------------------6

3.3 Data Preprocessing-------------------------------------------------------------6

3.4 Data Clustering------------------------------------------------------------------6

3.5 Get best model of each cluster----------------------------------------------6

3.6 Hyperparameter Tuning-------------------------------------------------------6

3.7 Model saving---------------------------------------------------------------------7

3.8 Cloud setup-----------------------------------------------------------------------7

3.9 Push App to cloud--------------------------------------------------------------7

3.10 Data from client side for prediction---------------------------------------7

3.11 Export prediction to CSV-----------------------------------------------------7

3.12 User Interface-------------------------------------------------------------------7

Unit Test Case---------------------------------------------------------------------------8

# **1. Introduction**

# ---------------

## 1.1 Low-Level Design Document Overview

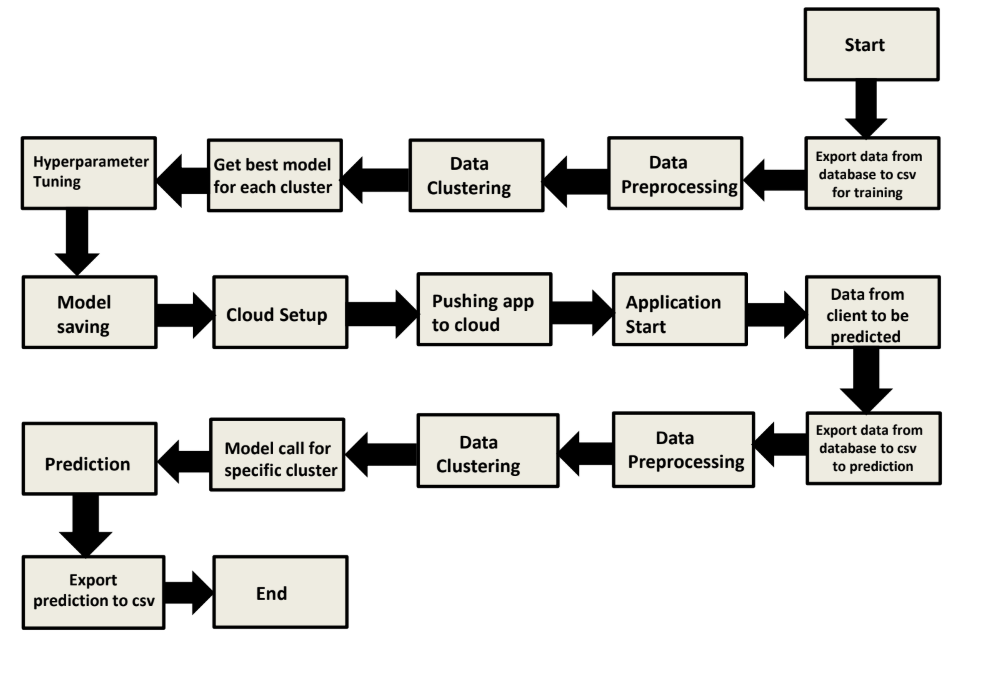
# 

This Low-Level Design (LLD) document provides a detailed description of the internal logical design of the Thyroid Disease Detection System. The LLD outlines the class diagrams, methods, and relationships between classes, as well as program specifications.

## 1.2 Scope

The Low-Level Design process is a component-level design approach that follows a step-by-step refinement process. This process encompasses the design of data structures, software architecture, source code, and performance algorithms. The data organization is defined during requirement analysis and refined during data design work.

# **2. Architecture**



# **3. Architecture Overview**

#### **3.1 Data Description**

For this project, we will utilize the Thyroid Disease Dataset available in the UCI Machine Learning Repository. This dataset meets our data requirements and consists of 7,200 instances distributed across various batches. The dataset includes both categorical and numerical attributes, making it suitable for comprehensive analysis and model training.

#### **3.2 Exporting Data for Training**

After data validation, the batch files are stored in a database. We will develop a module to export these validated datasets into a single CSV file, consolidating the data for training purposes. This step is essential to prepare the dataset for model development.

#### **3.3 Data Preprocessing**

In this phase, we will conduct an exploratory data analysis (EDA) to better understand the dataset. Based on this analysis, we will perform necessary data preprocessing, such as handling missing values through imputation, removing irrelevant columns, and other data cleansing tasks. A separate module will be created for this process, which will be applied both during the training phase and when making predictions on new data.

#### **3.4 Data Clustering**

To group similar data points, we will apply the K-Means clustering algorithm on the pre-processed dataset. The optimal number of clusters will be determined using the elbow method. The purpose of clustering is to segment the data into different groups, each of which will be trained with a specific model. After clustering, we will save the trained K-Means model for future use in predictions.

#### **3.5 Model Selection for Each Cluster**

For each cluster obtained in the clustering step, we will train multiple models and evaluate their performance to identify the best-performing model for each specific cluster. This approach ensures that each segment of data is handled by the most suitable model.

#### **3.6 Hyperparameter Tuning**

Once the best models are selected for each cluster, we will perform hyperparameter tuning to optimize the performance of each model. This tuning will involve adjusting key parameters to improve model accuracy. After tuning, the optimized models will be saved for future use.

#### **3.7 Model Storage**

Following the hyperparameter tuning, the final models for each cluster will be saved with corresponding cluster identifiers. These models will be used for making predictions on new client data, with the correct model being selected based on the client's data cluster.

#### **3.8 Cloud Setup**

We will prepare the necessary cloud infrastructure for deploying the application. This includes setting up all files and configurations required for cloud deployment, creating a Flask-based application, developing the user interface (UI), and integrating the trained models with the Flask app and UI.

#### **3.9 Application Deployment to Cloud**

Once the cloud setup is complete and the app functions as expected locally, the application will be deployed to cloud platforms such as Heroku and AWS. This step will ensure that the application is accessible online for users to access and utilize for predictions.

#### **3.10 Client-Side Data for Prediction**

When the application is deployed, it will be capable of receiving client-side prediction data. This data will be exported from the database, processed similarly to the training data (through modules created during training), and undergo data preprocessing, clustering, and model selection. Based on the data's cluster, the appropriate pre-trained model will be used to make predictions.

#### **3.11 Exporting Predictions to CSV**

After the predictions are generated for the client data, they will be exported to a CSV file. This file will then be delivered to the client, providing the results of the prediction process.

#### **3.12 User Interface Design**

#### **3.12.1 UI Overview**

The user interface for the Thyroid Disease Detection system will consist of three main pages designed to facilitate bulk prediction and display results efficiently.

#### **3.12.2 Index Page**

The Index page serves as the entry point for the application. It will provide users with options to upload data for bulk prediction or access other functionalities, including navigation to the result page.

#### **3.12.3 Result Page**

The Result page will display the outcomes of the bulk prediction process. There will be two primary actions available on this page:

1. **Download Result (Bulk)**: This button allows the user to download the full set of prediction results in CSV format.
2. **View Predicted Results**: This button provides users with the option to view the predicted results directly on the page.

Additionally, the predicted results will be displayed on this page, providing users with immediate insights into the predictions made by the system.

# **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 2.Application  is deployed | The Application should load  completely for the user when the URL is accessed |
| Verify whether the User is able to sign up in the application | 1. Application is  accessible | The User should be able to sign up in the application |
| Verify whether user is able to  successfully login to the application | 1. Application is  accessible  2.User is signed up to the application | User should be able to successfully login to the application |
| Verify whether user is able to see input fields on logging in | 1. Application is  accessible  2.User is signed up to the application 3.User is logged in to the application | User should be able to see input fields on logging in |
| Verify whether user is able to edit all input fields | 1.Application  is accessible  2.User is signed up to the application 3.User is logged in to the application | User should be able to edit all input fields |
| Verify whether user gets Submit  button to submit the inputs | 1. Application is  accessible  2.User is signed up to the application 3.User is logged in to the application | User should get Submit button to submit the inputs |
| Verify whether user get prediction/output back after submitting the inputs | 1. Application is  accessible  2.User is signed up to the application 3.User is logged in to the application | Users should get their output after submitting the inputs. |
| Verify whether the output which user gets is according to inputs user made. | 1. Application is  accessible  2.User is signed up to the application 3.User is logged in to the application | The output should be in accordance with the inputs the user made. |
| Verify whether users have the option to download their result or not. | 1. Application is  accessible  2.User is signed up to the application 3.User is logged in to the application | Users should have the option to download their output result. |