# Thyroid Disease Detection

## Low Level Document (LLD)

| | |
|---|---|
| **Written By** | **Ahmad Taquee, Priyanka Gupta, Diksha Sharma** |
| **Document Version** | **1.0** |
| **Last Revised Date** | **22-October-2021** |

# Document Control

## Change Records:

| Version | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 22-Oct-2021 | Ahmad Taquee, Priyanka Gupta, Diksha Sharma | Introduction & Architecture defined |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## Reviews:

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

## Approval Status:

| Version | Review Date | Reviewed By | Approved By | Comments |
|---|---|---|---|---|
| | | | | |

**Low Level Document**

# Contents

# 1. <u>Introduction</u>

## 1.1. What is Low-Level design document?

The goal of LLD or a low-level design document (LLD) is to give the internal logical design of the actual program code for Thyroid Disease Detection System. LLD describe the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

## 1.2. Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

# 2. <u>Architecture</u>
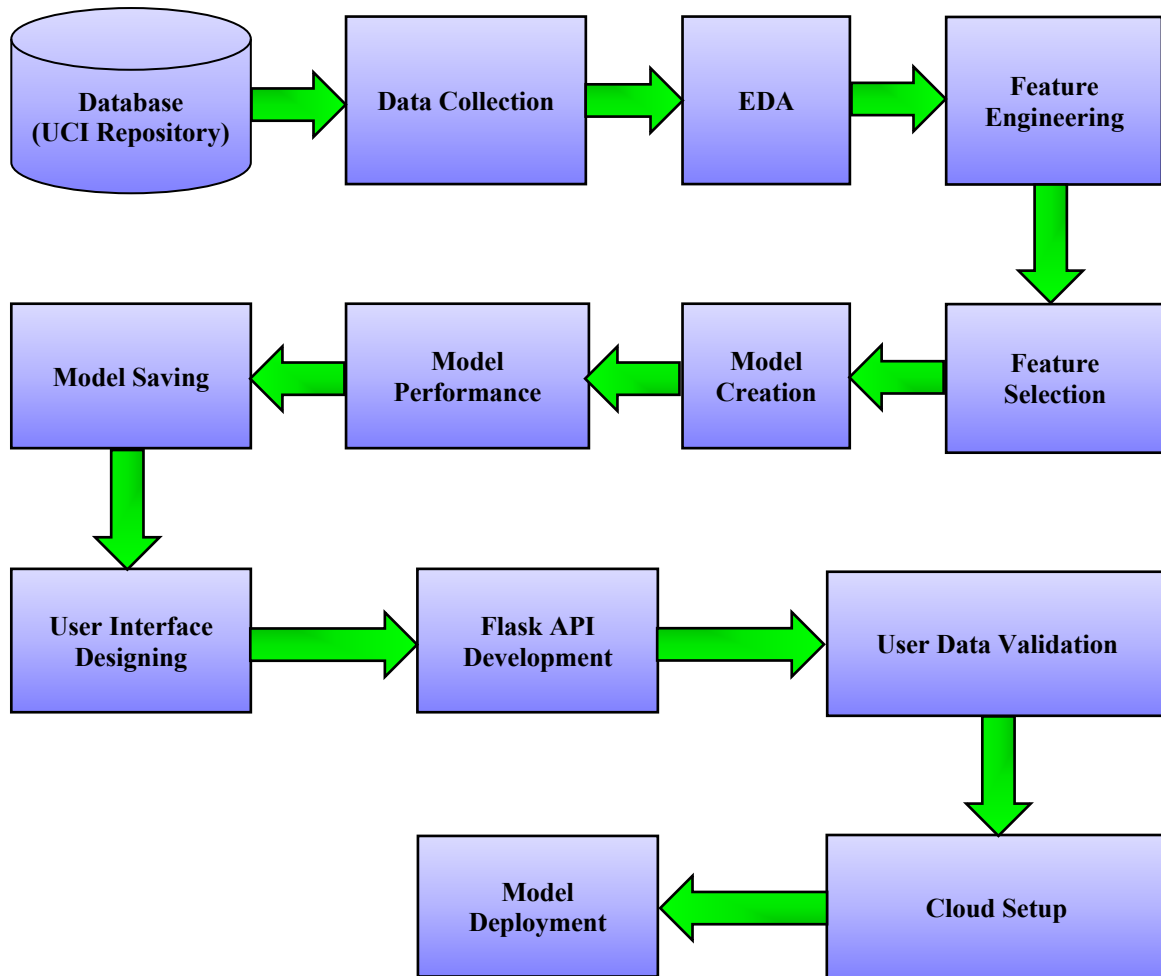
# 3. <u>Architecture Description</u>

## 3.1    Database (UCI Repository)

We are using two Thyroid Disease Data Sets present in UCI Machine Learning Repository which are **allhyper** dataset and **allhypo** dataset given in csv file formats. Allhyper dataset is to classify the records for hyperthyroid, T3 toxic, goitre, secondary toxic & negative classes whereas allhypo dataset is to classify the records for hypothyroid, primary hypothyroid, compensated hypothyroid, secondary hypothyroid and negative classes. Both have same records and columns, but classes are different.

Below is the link to access the datasets present in UCI repository:

https://archive.ics.uci.edu/ml/machine-learning-databases/thyroid-disease/

There are 2800 instances and 29 attributes in both the datasets.

Columns (same in both) present in both datasets are:

age, sex, on thyroxine, query on thyroxine, on antithyroid medication, sick, pregnant, thyroid surgery, I 131 treatment, query hypothyroid, query hyperthyroid, lithium, goitre, tumour, hypopituitary, psych, TSH measured, TSH, T3 measured, T3, TT4 measured, TT4, T4U measured, T4U, FTI measured, FTI, TBG measured, TBG, referral source, labels.

## 3.2    Data Collection

We imported both the csv files in Jupyter notebook using pandas library and concatenated the datasets as both are having same features. We also changed the labels as per the data records present in both the files using the logic to make only column as label column to be classified. This final label has 3 categories- Hyperthyroid, Hypothyroid and Negative.

## 3.3    EDA

In the Cleaning process, we cleaned up all the data which is not present in format which can be recognized by machine learning algos. Also, dropped the features which are not relevant. We handled the Null values and replaced with median value. We visualized the data using different libraries like seaborn, matplotlib, etc.

## 3.4    Feature Engineering

We replaced the outlier with median which was present in the age feature in the dataset. Also performed One hot and label encoding to convert the categorical features to numerical features. We transformed the data using different transformations like 'log' and 'square root' transformations to remove the skewness of the data present in different features.

## 3.5    Feature Selection

We dropped the columns which were not required after transforming the data. We also checked the multicollinearity between the independent features using correlation and VIF techniques. Highly correlated features were removed after this process.

## 3.6    Model Creation

After completing the feature selection process, data was split between train and test data. Dataset was imbalanced having high no. of negative classes in comparison to other classes therefore used SMOTE technique to handle it. Different machine learning algorithms like Logistic Regression, DecisionTree, SVM, RandomForest, K-NN were applied on the training data and testing data.

## 3.7    Model Performance

After fitting the data on different models, we checked the performance using different metrics like accuracy score, precision, recall. We got accuracy 99.92% on training data and 97.02% on test data.

## 3.8    Model Saving

After comparing all accuracies, we have chosen RandomForest as our best model and so we have dumped this model in a pickle file format.

## 3.9    User Interface Designing

We created a user interactive page where user can enter their input values to different fields of our application. In these front-end page, we designed a form which is taking values using text-boxes. These html user input data is transferred in json format to backend. After filling the fields values, user can see the result by clicking on the predict button.

## 3.10   Test Case Validation

Data validation is done here which was entered by user.

## 3.11   Cloud Setup

We are using Heroku cloud to deploy our application.

## 3.12   Model deployment

We deployed our model to Heroku which generate one URL. Using that URL, user can access the User Interface of the application.

# 4. <u>Unit Test Cases</u>

| Test Case Description | Pre-requisite | Expected Result |
|---|---|---|
| Verify whether the Application URL is accessible to the user. | 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 | Application URL is accessible. Application is deployed. | The Application should load completely for the user when the URL is accessed. |
| Verify whether user is giving standard input. | Handled test cases at backends. | User should be able to see successfully valid results. |
| Verify whether user is able to edit all input fields. | Application is accessible. | User should be able to edit all input fields. |
| Verify whether user gets Predict button to submit the inputs | Application is accessible. | User should get Submit button to submit the inputs. |
| Verify Predicted result is displayed. | Application is accessible. | User should be able to see the predicted result. |