Low Level Design

Thyroid Disease Detection Project

**Document Version**

* **Version**: 1.0
* **Last Revised Date**: 09-Nov-2024

**Change Record**

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09-Nov-2024 | [Your Name] | Initial draft for Thyroid Disease Detection LLD |

**Contents**

1. Introduction
2. What is Low-Level Design?
3. Scope
4. Architecture
5. Architecture Description
6. Data Description and Preprocessing
7. Model Training and Evaluation
8. API Design and Endpoints
9. Streamlit Frontend Design
10. Error Handling and Logging
11. Performance Metrics
12. Constraints and Assumptions
13. Deployment
14. Unit Test Cases

**1. Introduction**

The Thyroid Disease Detection project aims to predict thyroid-related conditions using patient data and medical indicators. This LLD document provides technical details, implementation specifics, and module interactions.

**2. What is Low-Level Design?**

The Low-Level Design (LLD) provides an in-depth breakdown of the technical aspects, including class diagrams, data flows, and interactions between components. It helps programmers implement the system accurately based on architectural guidance.

**3. Scope**

This LLD outlines component-level design for developing, testing, and deploying the thyroid disease prediction model and its API and web interface. The focus includes data preprocessing, model training, API development, and frontend implementation.

**4. Architecture**

* Data Input: JSON or Excel file
* Processing: Data preprocessing, feature engineering
* Model: Random Forest for thyroid disease classification
* API: Flask for model deployment
* Frontend: Streamlit for file uploads and result display

**5. Architecture Description**

The architecture includes components for data ingestion, preprocessing, model training, deployment, and API interaction with the frontend. Below are the details of each stage.

**6. Data Description and Preprocessing**

* Data: Patient attributes like age, sex, thyroid history, and lab results.
* Preprocessing Steps:
  + Handle missing values (imputation).
  + Encode categorical variables.
  + Scale continuous features (TSH, T3, etc.).
  + Feature selection for optimal model performance.

**7. Model Training and Evaluation**

* Model: Random Forest Classifier.
* Hyperparameter Tuning: Optimize with GridSearchCV.
* Metrics: Accuracy, F1-score, ROC-AUC.
* Model Storage: Saved with pickle or joblib.

**8. API Design and Endpoints**

* /predict (POST): Accepts JSON and returns prediction.
* /status (GET): Returns API status.
* Input Validation: JSON schema verification for required fields.

**9. Streamlit Frontend Design**

* File Upload: Supports JSON and Excel.
* Result Display: Shows disease prediction.
* Go Back Button: Allows new uploads by clearing previous results.

**10. Error Handling and Logging**

* Input Validation: Ensure correct data format.
* Logging: Python logging module for API requests, errors, and predictions.

**11. Performance Metrics**

* Model Accuracy: Target 90%+ accuracy.
* API Latency: Response time < 2 seconds.
* Memory Usage: Optimize with joblib.

**12. Constraints and Assumptions**

* Input Data: JSON/Excel files with required fields.
* Single Machine Execution: Local deployment limits scalability.

**13. Deployment**

* Environment: Local initially; future cloud deployment (e.g., AWS).
* Containerization: Docker for consistent deployment.

**14. Unit Test Cases**

| Test Case | Expected Outcome |
| --- | --- |
| API status check | API should return "running" status. |
| File upload (valid JSON) | Prediction result is displayed. |
| File upload (invalid format) | Error message returned. |

This LLD serves as a blueprint for developers and testers, covering the implementation, validation, and deployment of the Thyroid Disease Detection project.