HIGH LEVEL DESIGN (HLD)

THYROID DISEASE DETECTION

**High-Level Document for Thyroid Disease Detection Project**

**CONTENT**

1. Introduction 3

2. Objectives 3

3. System Architecture 3

4. Data Flow 5

5. Model Evaluation 5

6. Technologies and Tools 6

7. Deployment Strategy 7

8. Error Handling and Logging 7

9. Performance 8

10. Constraints 9

11. Future Work 9

12. Conclusion 10

13. Reference 10

1. Introduction

A High-Level Document (HLD) provides a comprehensive overview of a project, focusing on its architecture, key components, and overall design without going into intricate technical details. The scope of this document is to offer a bird’s eye view of the Thyroid Disease Detection project, laying out its objectives, architecture, tools, error handling, and more.

Scope:

- Summarize the project structure and goals.

- Outline system architecture and tools used.

- Highlight error handling, performance measures, and constraints.

- Serve as a reference document for stakeholders, developers, and non-technical readers.

2. Objectives

- Develop a robust machine learning system to predict thyroid diseases from patient data.

- Provide a user-friendly interface for data input and result visualization.

- Ensure scalability and ease of deployment for future enhancements.

3. System Architecture

3.1 Data Input

- Data comes from JSON or Excel files, containing medical and demographic information.

3.2 Backend (Machine Learning)

- Model: Random Forest Classifier.

- Features: Age, sex, thyroid history, lab tests (TSH, T3, etc.), and other clinical indicators.

- Target: Thyroid disease classification.

3.3 API and Web Interface

- Flask API: Serves the machine learning model and handles JSON-based input.

- Streamlit: A web interface that allows users to upload patient data files and view predictions.

4. Data Flow

1. User Input: Patient data is uploaded via JSON or Excel.

2. Processing: The Flask API accepts the data, applies the Random Forest model, and generates a prediction.

3. Output: Predictions are sent back to the Streamlit app for display.

5. Model Evaluation

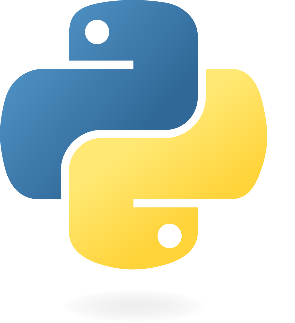
Evaluation Metrics: Accuracy, precision, recall, F1-score, ROC-AUC.

Validation: K-fold cross-validation ensures that the model performs consistently across different data subsets.

6. Technologies and Tools

6.1 Programming Languages

Python: For machine learning, API development, and data processing.



6.2 Libraries

- pandas, numpy: Data manipulation.

- scikit-learn: Model training and evaluation.

- pickle, joblib: Model serialization.

- Flask: API framework.

- Streamlit: Web interface for data upload and visualization.

6.3 Development Tools

- IDE: Spyder or Jupyter Notebook for development, VS Code for API and web app development.

- Version Control: Git for source code management.

7. Deployment Strategy

- Local Deployment: Initially designed to run on a local server for demo purposes.

- Future Consideration: Potential deployment on cloud platforms like AWS or Heroku for broader access.

- Docker: For containerizing the API and app for consistent deployments.

8. Error Handling and Logging

8.1 Error Handling

Input Validation:

- Ensure that uploaded files are in the correct format (JSON/Excel).

- Validate required fields in the data (e.g., `age`, `sex`, `TSH`).

- Handle missing values by applying pre-defined data imputation strategies or rejecting invalid records.

API Errors:

- If the Flask API encounters issues (e.g., incorrect input format), return standardized error messages like "400 Bad Request" or "500 Internal Server Error".

- Add try-except blocks around critical functions to handle runtime errors gracefully.

8.2 Logging

Model Predictions: Log input data and predicted output for debugging and audit purposes.

Errors: Capture errors and exceptions in log files with details like timestamps, error types, and stack traces.

API Requests: Log request metadata (IP address, timestamps, payload) for security and troubleshooting.

Tools Used for Logging:

- Python’s `logging` module for storing logs.

- Error monitoring tools (e.g., Sentry) can be integrated in future for real-time monitoring.

9. Performance

9.1 Model Performance

Training Accuracy: 90-95% (to be fine-tuned).

Precision: Focus on minimizing false positives in detecting thyroid diseases.

F1 Score: Balances precision and recall for handling imbalanced classes.

9.2 Computational Performance

Execution Time: The time taken for model inference is optimized to return results within seconds.

Memory Usage: Efficient memory management through joblib when loading large models.

10. Constraints

10.1 Data Constraints

Imbalanced Dataset: Thyroid disease data often contains an imbalance between positive and negative classes, affecting model performance.

Data Quality: Missing or incorrect data can reduce the accuracy of predictions.

10.2 Technical Constraints

Local Execution: The app currently runs on a local server, limiting scalability until cloud deployment.

Real-time Data: The current system works in batch mode and does not yet support real-time data inputs.

10.3 Model Constraints

Limited Explainability: Although the Random Forest model performs well, its decisions are not easily interpretable, which may be a concern in medical diagnostics.

11. Future Work

Advanced Model Development: Explore more sophisticated models, such as ensemble methods or deep learning, to improve accuracy.

Cloud Deployment: Deploy the system on a scalable cloud platform like AWS.

Explainability: Implement interpretability techniques like SHAP or LIME to make the model’s decisions more understandable.

Integration: Integrate the system with healthcare databases for real-time predictions.

12. Conclusion

The Thyroid Disease Detection project provides a machine learning-based approach for identifying thyroid-related issues using patient data. With its scalable architecture, user-friendly interface, and flexible deployment options, the project is poised to be a valuable tool in the healthcare domain. Future iterations will focus on improving model accuracy, explainability, and scalability, while refining the user experience.

This high-level document provides a comprehensive yet accessible overview, ensuring clarity on the project’s scope, architecture, and operational aspects for both technical and non-technical stakeholders.

13. Reference