HINDUSTAN AERONAUTICS LIMITED

KORAPUT DIVISION, ODISHA

Project Report On

LAB Report API

Submitted By:

Om Bhanuvilas Suryawanshi (25-32, CSE)

Sevella Vishal Reddy (25-34, AI&DS)

Sanskruti Rout (25-29, MECH)

Under The Guidance of

Mr. Laxmi Kanta Khillo

Sr. Manger (IT) ED.

Mr. Swadesh Behera

Sr. Manager (IT) ED.

Submitted To

TRAINING AND DEVELOPMENT INSTITUTE

HAL, KORAPUT

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude to Hindustan Aeronautics Limited, Engine Division, Koraput, Odisha having accorded approval to undertake this technical report in their organization.

I acknowledge my sincere thanks to Mr Laxmi Kanta Khillo, Sr. Manager (IT) and Swadesh Behera Sr Manager (IT) for their cooperation and guidance during my vocational training.

Last but not the least, I would like to express my sincere and humble gratitude to all the staff of IT Dept. for their cooperation and help at various stages to make this project a success.

Om Bhanuvilas Suryawanshi

Sevella Vishal Reddy

Sanskruti Rout

CERTIFICATE

This is to certify that Om Bhanuvilas Suryawanshi from KIIT University, Bhubaneshwar has successfully undergone one-month vocational training in HAL, Engine Division, Koraput, Odisha from 15th May to 14th June.

He has taken keen interest in learning and completing the vocational training in HAL. During his training period, he was familiarized with various activities of the IT department.

I found him to be sincere & hardworking and his performance was very good during the training period. His character and conduct were satisfactory.

I wish him all the success in his future endeavours.

Mr. Laxmi Kanta Khillo
Sr. Manger (IT)
Engine Division

Mr. Swadesh Behera
Sr. Manager(IT)
Engine Division

CERTIFICATE

This is to certify that Sevella Vishal Reddy from KL University Vijayawada has successfully undergone one-month vocational training in HAL, Engine Division, Koraput, Odisha from 15th May to 14th June.

He has taken keen interest in learning and completing the vocational training in HAL. During his training period, he was familiarized with various activities of the IT department.

I found him to be sincere & hardworking and his performance was very good during the training period. His character and conduct were satisfactory.

I wish him all the success in his future endeavours.

Mr. Laxmi Kanta Khillo
Sr. Manger (IT)
Engine Division

Mr. Swadesh Behera
Sr. Manager(IT)
Engine Division

CERTIFICATE

This is to certify that Sanskruti Rout from NIT Rourkela has successfully undergone one-month vocational training in HAL, Engine Division, Koraput, Odisha from 15th May to 14th June.

She has taken keen interest in learning and completing the vocational training in HAL. During his training period, she was familiarized with various activities of the IT department.

I found her to be sincere & hardworking and his performance was very good during the training period. Her character and conduct were satisfactory.

I wish her all the success in her future endeavours.

Mr. Laxmi Kanta Khillo
Sr. Manger (IT)
Engine Division

Mr. Swadesh Behera
Sr. Manager(IT)
Engine Division

CONTENTS

HISTORY OF HAL

Hindustan Aeronautics Limited (HAL) traces its origins to December 1940 when industrialist Seth Walchand Hirachand, in collaboration with the princely state of Mysore, established the original company. Registered on 23rd December 1940, HAL began by manufacturing Harlow Trainers, Curtiss Hawk fighters, and Vultee bombers in partnership with Intercontinental Aircraft Company (USA). The first aircraft was delivered to the Government of India in August 1941, along with a glider designed by Dr. V. Ghatage.

Licensing and Expansion (1960s–1980s)

In August 1962, India signed an agreement with the Soviet Union to manufacture MiG-21FL aircraft under license. To support this:

- Aero Engine Factory was set up in Koraput, Odisha
- Airframe Factory in Nasik, Maharashtra
- Avionics Factory in Hyderabad (then Andhra Pradesh)

To consolidate these efforts, Aeronautics India Limited was formed in April 1964, later merged into Hindustan Aeronautics Limited.

Government approval for the Sunabeda (Koraput) Aero Engine Factory came in March 1964, with the first R11F2-Series-III engines manufactured in 1969 and the first fully indigenous engine in 1971. The facility also began producing castings and forgings for MiG aircraft.

Subsequent agreements with the USSR led to:

- Manufacturing of MiG-21BIS (1976), powered by R25 engines. Production started in 1978-79; the first indigenous engine was completed in January 1983.
- Manufacturing of MiG-27M (1982), using R-29B engines, with production beginning around 1984-85.

To reduce dependence on Soviet suppliers, HAL began indigenizing raw materials and spare parts starting in 1977-78.

Corporate Objectives

Outlined in its Memorandum of Association, HAL's objectives include:

- Design, development, manufacture, repair, and overhaul of aircraft, engines, missiles, and related equipment.
- Enhancing self-reliance in defence aviation to meet evolving military and national needs.

In April 1971, a Review Committee was formed to define HAL's core objectives, later adopted by the Board in 1972 and approved by the Government in 1973. These include:

- Achieving self-sufficiency in aircraft and engine production.
- Encouraging professional growth, innovation, and national economic contribution.
- Fostering an adaptable, efficient, and committed workforce.

The objectives were reinforced in 1983 with added focus on:

- High-quality, reliable products.
- Strong after-sales support.
- Optimized capacity utilization and export potential.
- Increased indigenization of materials and products.

HAL operates through five major complexes across India:

1. Bangalore Complex
    - Aircraft Division: Jaguar aircraft

- Engine Division: Jaguar engines
- Helicopter Division: Various helicopters
- Forge & Foundry: Precision castings/forgings
- Overhaul Division: Aircraft and engine servicing
- Space Division: Satellite launch systems
- Servicing Division: Support services

2. MiG Complex
    - Nashik Division: Airframe manufacturing and overhaul
    - Koraput Division: Engine manufacturing for MiG & SU-30
3. Accessories Complex
    - Hyderabad: Electronics and avionics.
    - Kanpur: Passenger aircraft and gliders.
    - Lucknow: Hydraulic/fuel pumps, stator generators.
    - Korwa: Advanced navigation equipment.
4. Design Complex
    - Bangalore: Component/unit modification and R&D.
5. Helicopter Complex
    - Barrackpore: Helicopter components manufacturing and overhaul.

KORAPUT ENGINE DIVISION

The Engine Division at Koraput, a vital unit of Hindustan Aeronautics Limited (HAL), was established in April 1964 under an inter-governmental agreement with the Soviet Union (erstwhile USSR). Its initial mandate was the licensed manufacture of the R11-F2 turbojet engine for the MiG-21FL aircraft.

Over the years, the division expanded its capabilities under multiple licensing agreements to include:

- R11 Series Engines for MiG-21FL and MiG-21M aircraft
- R25 Series Engines for MiG-21BIS aircraft
- R29B Engines for MiG-27M aircraft

To support lifecycle maintenance, dedicated overhaul facilities for these engines were established:

- Overhaul of R11 series began in 1971
- Overhaul of R25 series started in 1983
- From 1997-98, the division also undertook overhaul of RD-33 engines used in MiG-29 aircraft

Departments in Engine Division – Koraput

The division comprises a wide range of specialized departments to support engine manufacturing, repair, and overhaul operations:

XII. Forge
XIII. Foundry
XIV. Tool Room
XV. Small Parts & Fuel
XVI. Sheet Metal & Welding
XVII. Blades
XVIII. Electroplating
XIX. Heat Treatment

XX. Compressor
XXI. Turbine
XXII. CNC Machine
XXIII. Assembly
XXIV. Overhaul
XXV. Gear
XXVI. Test House
XXVII. Maintenance

Sukhoi Engine Division has 6 workshops, called as Hangar.

Each hangar has different work for engine.

1.HANGER 101

- Compressor Shop-611
- Turbine Shop-610
- Blade Shop-615

2.HANGER 102

- CNC Shop-616
- Gear Casing Shop-609

3.HANGER 103

- Heat Treatment Shop-603
- Electroplating Shop-604

4. HANGER-104

- Assembly-620
- Jet Nozzle Assembly-620A
- Fuel&Small Parts-614
- Rig Room-614A

5.HANGER-105

- Sheet Metal Shop-612
- Special Equipment Shop

6.HANGER-106

- Aggregate Overhaul Shop-613

SL.NO	AIRCRAFT	ENGINE	INDIGENOUS NAME
1	MIG-21FL	R11-F20	BADAL
2	MIG-21M/MF	R11-F2S/F2SK	TRISHUL
3	MIG-21BIS	R-25	VIKARM
4	MIG-23MF	R-29	RAKSHAK
5	MIG-23BN	R-29B	VIJAY
6	MIG-25	R-29B	GARUD
7	MIG-27M	R-29B	BAHADUR
8	MIG-29	RD-33	VAJ
9	SUKHOI 30MKI	AL-31FP	FLANKER-H
10	GNAT	ORPHEUS-701	AJEET
11	HF-24	ORPHEUS-703	MARUT
12	HJT-16	VIPER-11	KIRAN
13	JAGUAR	ADOUR MK-803	SHAMSHOR
14	MIRAGE-2000	M-53	VAJRA
15	LCA MK1A	GE F404-IN20	TEJAS
16	HS-748	DART-531	CHITRA
17	ALLOUTEE	ARTOUSTE-IIIB	CHEETAH

Project- LAB Report API

GitHub Link- https://github.com/Om-Suryawanshi/LabReportAPI

In many diagnostic laboratories, the process of collecting, recording, and transferring lab report data is still manual and outdated. The traditional workflow typically involves lab assistants collecting patient samples and running diagnostic tests using machines that display the results on-screen. These results are manually transcribed into a physical logbook and, later, entered into the hospital management system (HMS).

This method, while simple, presents several critical challenges:

- Prone to Human Error: Manual transcription of lab results increases the risk of data entry mistakes, which can lead to misdiagnosis or delays in treatment.
- Time-Consuming: Double entry (once into a logbook and then into HMS) wastes valuable staff time and delays the availability of lab reports.
- Lack of Real-Time Access: Doctors and medical staff often face delays in accessing lab results due to the non-instantaneous data flow.
- No Digital Audit Trail: In the absence of automated logging, it's difficult to trace when and how data was recorded or modified.
- Security Risks: Sensitive patient data handled manually is more susceptible to loss, theft, or unauthorized access.
- No Backup Mechanism: Physical records are vulnerable to damage, and there is often no secure backup system in place.

Need for a New System

With the increasing reliance on technology in the healthcare sector, there is a strong need for a digitally automated, secure, and reliable system to collect, validate, and store lab data directly from diagnostic machines. Such a system must be designed to work within secure, air-gapped environments (like those at HAL) where traditional cloud or networked solutions are not viable.

Proposed Solution – LabReportAPI

To address the inefficiencies and risks of the traditional lab report workflow, the LabReportAPI system has been developed as a secure, automated, and offline-capable middleware solution. It bridges the gap between diagnostic machines and data storage systems, ensuring real-time, error-resistant, and traceable report handling—even in network-isolated environments like HAL facilities.

System Overview

The LabReportAPI functions as a background service on a middleware PC, directly connected to diagnostic machines over TCP/IP. Instead of relying on manual data entry, machines send lab data automatically via TCP, which is:

- Received
- Validated
- Cached
- Securely saved in structured JSON format on a USB device

This automation ensures that only valid and complete reports are stored, and every interaction (USB detection, data reception, validation status) is logged for auditability.

Why USB-Based Transfer?

In high-security environments like Hindustan Aeronautics Limited (HAL), systems are often isolated from the internet or external networks for cybersecurity reasons. This makes real-time cloud-based storage or network-based APIs unsuitable.

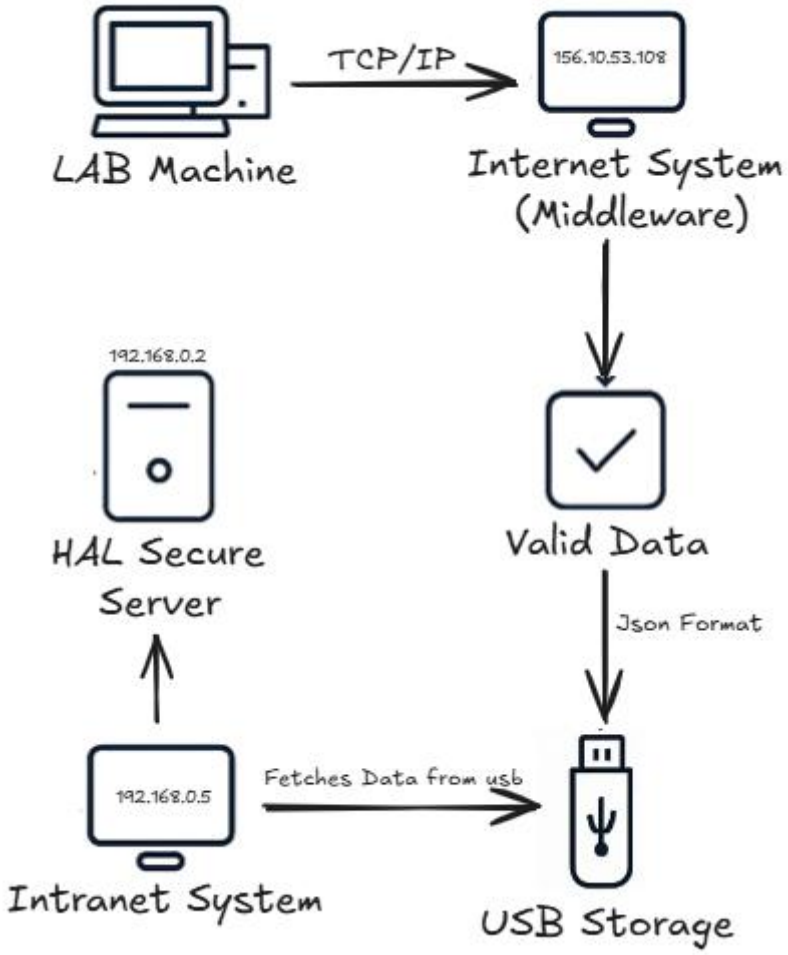
To tackle this, USB drives are used as secure physical bridges between:

- Untrusted zones (diagnostic machine systems, public networks), and
- Trusted zones (defence networks)

By using a removable, offline storage mechanism, the system maintains air-gap integrity while still enabling digital automation and traceability.

System Architecture

The LabReportAPI system is architected for reliability, modularity, and offline operability. It is designed to operate as a background TCP server, receiving real-time lab data from diagnostic equipment, validating it, and securely storing it to a USB device. The system also exposes a minimal set of API endpoints for monitoring and manual actions.



Data Flow Description

1. TCP Connection Initiation
    - Diagnostic machines establish a TCP connection with the server running LabReportAPI.

- The connection is received by a background service (TcpListenerService), which keeps listening on a specified IP and port.

2. Message Reception and Buffering
    - Raw message bytes are received and appended to an in-memory buffer.
    - Messages are structured with special start (STX: \x02) and end (ETX: \x03) markers.
3. Data Validation
    - The buffer is parsed to extract complete messages.
    - Each message undergoes rigorous checks to confirm:
        - Proper format
        - Absence of corruption or malicious content
        - Valid and non-empty fields
    - Appropriate response is sent back to the machine:
        - ACK (\x06) for valid data
        - NAK (\x15) for invalid or rejected messages
4. Message Caching and Auto-Save
    - Validated messages are stored in-memory.
    - If no new data is received for 30 seconds, the system writes the cached messages to a JSON file on the USB.
    - If a file already exists, the data is merged, not overwritten.
5. USB Scanning & File Handling
    - If the configured USB path is invalid or unavailable, the system:
        - Scans all connected drives.
        - Finds the first removable and ready USB drive

Implementation Details

The LabReportAPI system has been implemented using .NET 9 and follows a modular, service-based architecture. It uses the BackgroundService pattern to create a robust TCP listener that can handle multiple incoming connections and process data asynchronously.

TCP Listener Service

The heart of the application is the TcpListenerService, located in Services/TcpListenerService.cs. It inherits from BackgroundService and runs continuously in the background as long as the application is active.

Key Features of TcpListenerService:

- Listens for TCP clients on a specified IP address and port
- Accepts multiple client connections asynchronously
- Delegates each client to HandleClientAsync, ensuring the main loop remains non-blocking
- Gracefully stops on application shutdown to prevent data corruption

CODE SNIPPETS:

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken) {
var localEndPoint = new IPEndPoint(IPAddress.Parse(_localIp), Port);
_tcpListener = new TcpListener(localEndPoint);
_tcpListener.Start();
var saveTask = SaveMessagesToJsonPeriodically(stoppingToken);
while (!stoppingToken.IsCancellationRequested)
{
var tcpClient = await _tcpListener.AcceptTcpClientAsync(stoppingToken);
_ = HandleClientAsync(tcpClient, stoppingToken);
```

```
}
_tcpListener?.Stop();
}
private async Task HandleClientAsync(TcpClient client, CancellationToken ct) {
var clientEndpoint = client.Client.RemoteEndPoint?.ToString();
var clientIp = ((IPEndPoint)client.Client.RemoteEndPoint!).Address.ToString();
using (client)
            {
               client.SendTimeout = 5000;
               client.ReceiveTimeout = 5000;
               using var stream = client.GetStream();
               var buffer = new byte[BufferSize];
               var messageBuilder = new StringBuilder();
               while (!ct.IsCancellationRequested && client.Connected)
               {
                  var bytesRead = await stream.ReadAsync(buffer, 0, BufferSize, ct);
                  if (bytesRead == 0
                        break;
                  messageBuilder.Append(Encoding.UTF8.GetString(buffer, 0, bytesRead));
                  await ProcessMessageBuffer(messageBuilder, stream, ct, clientIp, client);
               }
            }
```

Data Validation Logic

Implemented in the ProcessMessageBuffer method, this component ensures that only clean, well-formed, and safe data is accepted and processed.

Validation Steps:

1. Receives a stream of bytes from the TCP client

2. Scans the buffer for:
    - Start of Text (STX = \x02).
    - End of Text (ETX = \x03).
3. Extracts and validates message content:
    - Checks for empty or malformed messages.
    - Filters out potentially malicious data.
4. Responds to the client with:
    - ACK (\x06) if the message is valid.
    - NAK (\x15) if the message is invalid or corrupted.

CODE SNIPPETS:

```
private async Task ProcessMessageBuffer(StringBuilder buffer, NetworkStream stream,
CancellationToken ct, string clientIp, TcpClient client) {

while (buffer.Length > 0)

        {

           var etxIndex = buffer.ToString().IndexOf('\x03');

           if (etxIndex ==-1) break;

           var stxIndex = buffer.ToString().IndexOf('\x02');

           if (stxIndex ==-1 || etxIndex <= stxIndex)

           {

              await SendResponse(stream, "\x15", ct); // NAK

              buffer.Clear();

              RegisterError(clientIp); // Don't block immediately

              return;

           }

           var message = buffer.ToString(stxIndex + 1, etxIndex- stxIndex- 1);

           buffer.Remove(0, etxIndex + 1);

           if (ContainsMaliciousContent(message))
```

```
            {
                await SendResponse(stream, "\x15", ct); // NAK
                RegisterError(clientIp);
                continue;
            }

            if (CheckRateLimit(clientIp))
            {
                RegisterError(clientIp);
                continue;
            }

            if (!ValidateLabMessage(message))
            {
                await SendResponse(stream, "\x15", ct); // NAK
                RegisterError(clientIp);
                continue;
            }
            await ProcessLabData(message);
            await SendResponse(stream, "\x06", ct); // ACK
        }
    }
private async Task SendResponse(NetworkStream stream, string response, CancellationToken ct)
    {
        var responseBytes = Encoding.UTF8.GetBytes(response);
        await stream.WriteAsync(responseBytes, 0, responseBytes.Length, ct);
    }
```

Data Storage Strategy

The application uses a background process (SaveMessagesToJsonPeriodically) that:

- Monitors for inactivity (no messages received for 30 seconds).
- Automatically writes buffered data to a USB drive in JSON format.

How it Works:

- Checks for a valid USB path from appsettings.json.
- If not found, scans all drives for the first ready removable drive.
- Merges new data with any existing JSON data to avoid overwrites.
- Clears the in-memory buffer after writing to disk.

Failure Handling:

- If no USB is found, logs a warning.
- Waits for 10 seconds before retrying.
- Keeps trying in the background until a USB is available.

CODE SNIPPETS:

```
private async Task SaveMessagesToJsonPeriodically(CancellationToken stoppingToken){
var idleTimeout = TimeSpan.FromSeconds(30);
        while (!stoppingToken.IsCancellationRequested)
        {
           var now = DateTime.UtcNow;
           var idle = (now- _lastReceivedMessage) > idleTimeout;
           if (idle && LabMessages.Count > 0)
           {
              _lastWriteStatus = "Writing to USB...";
```

```
                DriveInfo? usbDrive = null;
               if (!string.IsNullOrWhiteSpace(_settings.UsbPath))
                   {
                       usbDrive = new DriveInfo(_settings.UsbPath);
                       if (!usbDrive.IsReady || usbDrive.DriveType != DriveType.Removable)
                           usbDrive = null;
                   }
                   if (usbDrive == null)
                   {
                       usbDrive = DriveInfo.GetDrives().FirstOrDefault(d => d.DriveType ==
DriveType.Removable && d.IsReady);
                   }
                   if (usbDrive == null)
                   {
                       _lastWriteStatus = "USB not found";
                       await Task.Delay(TimeSpan.FromSeconds(10), stoppingToken);
                       continue;
                   }
                   var filename = "LabData.json";
                   var path = Path.Combine(usbDrive.RootDirectory.FullName, filename);
                   int retryCount = 3;
                   while (retryCount-- > 0)
                   {
                         var messagesSnapshot = LabMessages.ToArray();
                         List<LabMessage> allMessages = new();
                         if (File.Exists(path))
                         {
                            var existingJson = await File.ReadAllTextAsync(path, stoppingToken);
```

```
                    allMessages = JsonSerializer.Deserialize<List<LabMessage>>(existingJson) ??
new();
                }
                allMessages.AddRange(messagesSnapshot);
                var updatedJson = JsonSerializer.Serialize(allMessages, new JsonSerializerOptions {
WriteIndented = true });
                await File.WriteAllTextAsync(path, updatedJson, stoppingToken);
                while (!LabMessages.IsEmpty)
                    LabMessages.TryTake(out _);
                _lastWriteStatus = $"Saved at {now:HH:mm:ss}";
                _lastWriteTime = now;
                break;
            }
        await Task.Delay(TimeSpan.FromSeconds(10), stoppingToken);
      }
    }
public async Task<(bool success, SaveResult result)> TriggerManualSave()
    {
      var result = await SaveMessagesToUsb(force: true);
      return (result.StatusCode == "SUCCESS", result);
    }
```

API Specifications

The LabReportAPI exposes a minimal set of RESTful API endpoints that provide monitoring and control functionalities. These endpoints are primarily used for checking the system's status and manually saving data if required. They are useful for integration with a lightweight frontend or for administrative tasks.

GET /api/labdata/status

Purpose:
Returns the current state of the TCP Listener service—indicating whether it is running and accepting connections.

Use Case:
This endpoint is used by system administrators or frontend dashboards to verify if the middleware service is active and functioning properly.

Response:

{

"status": "Active",

"protocol": "STX/ETX",

"port": 12377,

"serverIp": "192.168.0.3",

"lastMessageReceivedAt": "2025-06-11T09:35:10.9395152Z",

"lastWriteStatus": "Manual save at 09:37:02",

"lastWriteTime": "2025-06-11T09:37:02.714519Z"

}

POST /api/labdata/save

Purpose:
Triggers the manual saving of collected lab data from the in-memory buffer to the USB device.

Use Case:
Useful when an immediate backup is needed without waiting for the 30-second inactivity timeout. This can also be triggered before scheduled maintenance or system shutdown.

Behavior:

- Validates USB availability
- Merges new messages with any existing data on the USB
- Clears the buffer after successful write

Response:


```
{
  "success": true,
  "statusCode": "SUCCESS",
  "message": "Manual save at 09:35:15",
  "messagesSaved": 3
}
```


GET /api/logs

Purpose:
Retrieves the log data generated by the system, including events like:

- USB detection
- Data reception

- Validation status
- Save events and errors

Use Case:
Allows administrators or developers to review activity logs for debugging, audits, or tracking system behavior over time.

Sample Response:


```
[
 {
  "timestamp": "2025-06-11T10:15:30Z",
  "event": "USB drive detected at E:\\",
  "type": "Info"
 },
 {
  "timestamp": "2025-06-11T10:16:12Z",
  "event": "Received valid message from 192.168.1.5",
  "type": "Success"
 },
 {
  "timestamp": "2025-06-11T10:17:45Z",
  "event": "Saved 10 messages to JSON file",
  "type": "Success"
 }
]
```

Security Considerations

Security is a fundamental aspect of the LabReportAPI system design. Since the middleware is intended for deployment in air-gapped environments like those used in defence and aerospace sectors (e.g., HAL), the system prioritizes offline security, data integrity, and controlled access over internet connectivity.

Network Isolation by Design

- The system does not depend on any external network or cloud service.
- All communication occurs over local TCP/IP connections between lab machines and the middleware PC.
- This makes the system highly resistant to external cyberattacks, malware, or unauthorized remote access.

Secure USB Handling

- USB drives are used as controlled physical transfer mechanisms between untrusted (test lab) and trusted (internal network) zones.
- Before writing data:
    - The system verifies the USB's readiness.
    - Ensures file handling is non-destructive (i.e., merges new data with existing JSON content).
- If a USB is not detected, the system logs a warning and safely retries after 10 seconds—ensuring no data is lost.

Data Validation and Input Sanitization

Before any incoming message is processed:

- The system checks for valid formatting using STX (\x02) and ETX (\x03) markers.
- Malformed or suspicious messages are:

    - Rejected
    - Logged
    - Responded to with a NAK (\x15)
- Valid messages receive an ACK (\x06), ensuring clear client-server communication.

This ensures that:

- No corrupted, malicious, or incomplete data makes it into the final report files.
- Each interaction is logged, providing a tamper-evident trail.

Minimal Attack Surface

- The application exposes only three API endpoints with no public internet exposure.
- These endpoints are used only for internal diagnostics and manual operations.
- No direct access to the USB filesystem is provided via the API.

Planned Future Enhancements

To further strengthen the system, the following features are proposed for future versions:

- USB Data Encryption: Encrypt JSON report files using symmetric or asymmetric encryption to prevent unauthorized viewing if the USB is misplaced.
- Checksum Validation: Add file checksums to detect tampering or accidental corruption during physical transfer.
- Role-Based Access: Implement restricted API access based on user roles for sensitive operations.

Testing and Validation

Thorough testing and validation were conducted to ensure that the LabReportAPI system performs reliably under expected operating conditions. The testing focused on data integrity, system responsiveness, fault tolerance, and correct behaviour in edge cases.

Unit Testing

Purpose: To validate the correctness of individual components, particularly the data processing and validation logic.

Tests Included:

- Message parsing (STX–ETX detection)
- Validation function for malformed data
- JSON merge functionality
- ACK/NAK response generation

Result:

All unit tests passed, confirming that isolated methods behave correctly for both valid and invalid input scenarios.

Integration Testing

Purpose: To validate the interaction between components including TCP reception, validation, storage, and logging.

Test Scenarios:

- TCP client sends a well-formed message → API stores it to USB
- TCP client sends corrupted or malformed message → System logs rejection
- USB unplugged during write → System retries after delay without crash

- Multiple clients sending data concurrently → Messages are properly queued and handled asynchronously

Result:
The system maintained stability, correctly processed messages, and handled errors gracefully across all tested scenarios.

Performance Under Load

Purpose: To assess the system's performance with high-volume or rapid message traffic.

Method:

- Simulated multiple clients sending continuous lab data over TCP
- Measured memory usage, write delay, and system responsiveness

Findings:

- The in-memory buffer handled large volumes without overflow
- JSON file writing was triggered correctly after 30 seconds of inactivity
- No significant performance degradation was observed during peak load

USB Handling Validation

Tested Scenarios:

- USB removed mid-process → Warning logged, system waited and retried
- USB not available → Data safely held in memory until USB is reconnected

The comprehensive testing approach confirms that the LabReportAPI is not only functional but also reliable and ready for deployment in critical, high-security environments.

Results and Observations

Following development and extensive testing, the LabReportAPI system demonstrated strong performance, operational efficiency, and reliability in real-world-like scenarios. The system successfully addressed the shortcomings of the traditional manual workflow and offered a secure, automated alternative for lab data management.

Elimination of Manual Entry

- The system entirely removed the need for lab assistants to manually note down or transcribe diagnostic results.
- This led to a significant reduction in human error and increased data accuracy.

Real-Time Data Capture

- Reports were received and processed in real time, ensuring that lab data was instantly available for transfer.
- No delays between test completion and data recording were observed.

Enhanced Security

- Using USB drives for air-gapped environments ensured data isolation without compromising transfer efficiency.
- Sensitive medical data remained protected, with no exposure to external networks.

Improved Efficiency and Workflow

- Automation of validation and storage processes freed up lab personnel, allowing them to focus on more critical tasks.
- A 30-second timeout-based auto-save mechanism ensured that data was written to disk in a timely but non-intrusive manner.

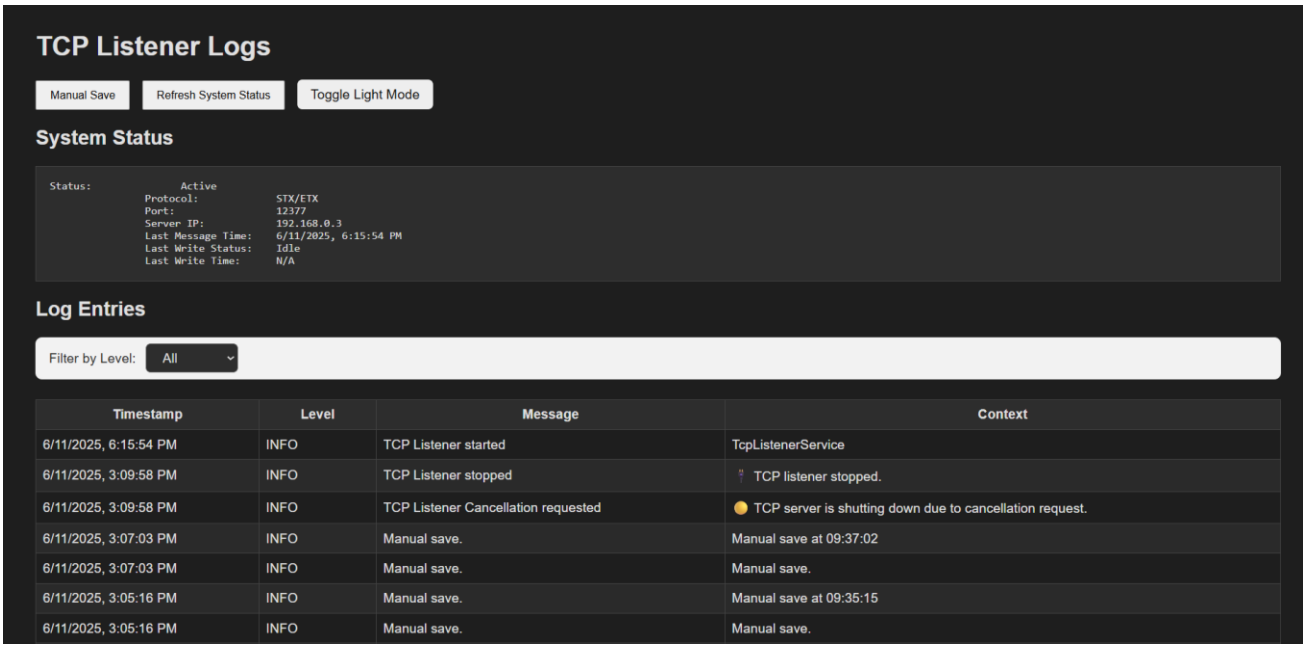
System Stability and Fault Tolerance

- Even when USB drives were disconnected or unavailable, the system continued to buffer data in memory and logged warnings without crashing.
- The ability to merge new data with existing JSON files prevented accidental data loss or overwrites.

Audit and Traceability

- All system events, including message validation results and storage actions, were logged.
- This provided a transparent audit trail, essential for quality assurance, compliance, and diagnostics.

Frontend Result on the “/” endpoint.



Future Enhancements

While the current version of LabReportAPI fulfils its intended objectives, several improvements have been identified to further strengthen the system's security, usability, and feature set. These enhancements aim to make the solution more robust, user-friendly, and production-ready for long-term deployment.

USB Encryption and File Integrity

- Objective: Protect sensitive lab data even if the USB drive is misplaced or stolen.
- Planned Features:
    - Implement AES encryption for JSON report files before saving to USB.
    - Add SHA-256 checksum generation to detect file tampering or corruption.
- Benefit: Ensures that report files remain confidential and tamper-evident outside the system.

Web-Based Analytics Dashboard

- Objective: Provide an intuitive interface for administrators to monitor system activity and analyse lab data trends.
- Planned Features:
    - Interactive dashboard to view logs, message counts, and USB write status.
    - Charts and filters for date-wise report visualization.
- Benefit: Enables real-time monitoring and supports decision-making with data insights.

Role-Based Access Control (RBAC)

- Objective: Restrict access to sensitive API endpoints and logs based on user roles.
- Planned Features:
    - Admin, Operator, and Read-only roles
    - Secure login mechanism for frontend tools
- Benefit: Adds a layer of access control, protecting system functions from unauthorized use.

Configurable Timeout & Storage Settings

- Objective: Allow system administrators to customize behaviour based on workflow.
- Planned Features:
    - Configurable idle timeout (default 30 seconds)
    - Optional automatic USB backup intervals
- Benefit: Improves flexibility and allows adaptation to different lab environments.

Conclusion

The LabReportAPI project was developed to address critical inefficiencies and security limitations in the traditional lab report workflow. Through a thoughtful combination of automated data collection, in-memory caching, robust validation, and secure USB-based storage, the system has successfully replaced manual processes with a more accurate, efficient, and auditable solution.

By leveraging a simple but powerful TCP/IP-based architecture, the middleware enables diagnostic machines to directly transmit lab data without human intervention. The system is tailored for air-gapped environments, such as those found in defence and aerospace industries, where cloud-based or networked alternatives are not feasible.

Key outcomes of the project include:

- Significant reduction in human error
- Reliable real-time data ingestion
- Strong data validation and logging mechanisms
- Secure, offline-first USB-based transfer
- A scalable and extendable .NET-based architecture

Moreover, the project sets a strong foundation for future improvements such as encryption, analytics dashboards, and role-based access control, making it a forward-compatible solution.

In conclusion, LabReportAPI is not just a middleware application—it is a step forward in modernizing laboratory workflows, enhancing data integrity, and ensuring secure report handling in high-security domains.

References

1. https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/sockets/tcp-classes

2. https://www.codeproject.com/Articles/5270779/High-Performance-TCP-Client-Server-using-TCPListen

3. https://stackoverflow.com/questions/38859872/writing-files-to-external-flash-drive-in-windows-universal

4. https://tempuzfugit.wordpress.com/2007/10/08/external-storage-unit-detection-with-c-in-net-usb-card-readers-etc/