**Software Requirements Specification (SRS)**

**Table of Contents**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, and Abbreviations
   4. References
2. Overall Description
   1. Product Perspective
   2. Product Functions
   3. User Classes and Characteristics
   4. Operating Environment
   5. Design and Implementation Constraints
   6. Assumptions and Dependencies
3. Specific Requirements
   1. Portal 1
   2. Portal 2
   3. Portal 3
4. External Interface Requirements
5. System Features
6. Non-functional Requirements
7. Other Requirements

### 1. Introduction

#### 1.1 Purpose

In order to create three interrelated online platforms for login capabilities, dashboard management systems, request forms, and maintenance alert systems, the purpose of this document is to provide a detailed description of the requirements.

#### 1.2 Scope

* **Portal 1**: Company Request Handling & Dashboard
* **Portal 2**: ISO-17025 Compliance with Templates and Forms
* **Portal 3**: Maintenance and Expiry Alerts for Gas Rooms

#### 1.3 Definitions, Acronyms, and Abbreviations

* **ISO**: International Organization for Standardization
* **SRS**: Software Requirements Specification
* **PDF**: Portable Document Format
* **jsPDF**: JavaScript library to generate PDF files.
* **UI**: User Interface

#### 1.4 References

* Documentation on ISO 17025 standard
* Documentation for jsPDF library

### 2. Overall Description

#### 2.1 Product Perspective

The product is a collection of three web portals which are interconnected in such a way that they form one system whereby different parts or aspects, such as request handling, documentation compliance, among others, may be performed.

#### 2.2 Product Functions

* **Portal 1**: Login; dashboards; forms for making requests and receiving them.
* **Portal 2**: ISO-17025 manual; employee logins; information on services/tests offered within specific laboratories; creation of document templates using which various types of records may be generated; creating forms for data input into computer applications then exporting such entries into PDF format through these said computer applications.
* **Portal 3**: Entries made about maintenance activities carried out on rooms where gas cylinders are kept; either automatically or manually generated alerts notifying concerned personnel about expiration dates for certain gases stored in such rooms combined with notices indicating reasons why it happened at what time, etc.

#### 2.3 User Classes and Characteristics

* Employees
* Administrators
* Maintenance personnel
* Managers

#### 2.4 Operating Environment

The system will operate on web browsers and be accessible via the internet.

#### 2.5 Design and Implementation Constraints

* Compliance with ISO-17025
* Secure login mechanisms
* Responsive design for various devices

#### 2.6 Assumptions and Dependencies

* Users will have internet access.
* Users are familiar with web interfaces.

### 3. Specific Requirements

#### 3.1 Portal 1: Request Handling

##### 3.1.1 Login Page

* **Description**: A secure login page for users.
* **Requirements**:
  + User authentication (username/password)
  + Password recovery option

##### 3.1.2 Dashboard

* **Description**: A user dashboard displaying relevant information and options.
* **Requirements**:
  + Overview of pending requests
  + Access to request forms
  + Notification panel

##### 3.1.3 Request from the Company - Landing Page + Form

* **Description**: A landing page with a form for company requests.
* **Requirements**:
  + Input fields for request details
  + Form submission button
  + Data validation

##### 3.1.4 Receiving Page for the Request

* **Description**: Page to view and manage received requests.

#### **Requirements**:

* + List of received requests
  + Option to approve or reject requests
  + Search and filter functionality

#### 3.2 Portal 2: ISO-17025 Compliance

##### 3.2.1 Manual in Drop-down Format

* **Description**: A drop-down menu to navigate the ISO-17025 manual.
* **Requirements**:
  + Hierarchical display of manual sections
  + Search functionality

##### 3.2.2 Employee Login Page

* **Description**: Secure login page displaying employee hierarchy.
* **Requirements**:
  + User authentication
  + Hierarchical display of user roles

##### 3.2.3 Landing Page (Services & Types of Test)

* **Description**: Landing page displaying available services and types of tests.
* **Requirements**:
  + List of services
  + Detailed information on each type of test

##### 3.2.4 Template of the Documents

* **Description**: Document templates available for download.
* **Requirements**:
  + 4/5 different templates
  + Download options

##### 3.2.5 Create Simple Forms

* **Description**: Form creation tool with template selection.
* **Requirements**:
  + Form builder with input fields
  + Template selection option
  + Data validation
  + Save form data to the database

##### 3.2.6 Export to PDF

* **Description**: Export form data to PDF using jsPDF.
* **Requirements**:
  + Convert form data to PDF
  + Download or email PDF option

#### 3.3 Portal 3: Entries & Maintenance

##### 3.3.1 Gas Room Expiry & Maintenance Levels

* **Description**: Track expiry and maintenance levels for gas rooms.
* **Requirements**:
  + Data entry for gas maintenance schedules
  + Alert system for nearing expiry/maintenance dates

##### 3.3.2 Reminding via Pop-up/Alert

* **Description**: Alert users when limits are reached.
* **Requirements**:
  + Pop-up alerts for maintenance/expiry
  + Email notifications

### 4. External Interface Requirements

* **User Interfaces**: Web pages for login, dashboards, forms, templates, and alerts.
* **Hardware Interfaces**: No specific hardware requirements beyond standard web access.
* **Software Interfaces**: Integration with database and jsPDF library.
* **Communication Interfaces**: HTTPS for secure data transmission.

### 5. System Features

* Secure authentication and authorization
* Responsive design
* User-friendly interfaces
* ISO-17025 compliance

### 6. Non-functional Requirements

* **Performance:**

A high-performing website ensures users don't have to wait long for pages to load. This is achieved through techniques such as optimizing images, **minimizing HTTP requests,** using a **content delivery network (CDN)**, and implementing efficient caching strategies. Fast load times keep users engaged and reduce bounce rates, while responsive interactions ensure smooth navigation and a seamless user experience.

* **Security:** Security is crucial for protecting user data and maintaining trust. Data encryption ensures that information transmitted between the user and the server is secure from interception. Implementing secure login processes, such as **two-factor authentication**, adds an extra layer of security. Role-based access control restricts access to sensitive areas of the website based on user roles, ensuring that only authorized individuals can access certain data or perform specific actions.
* **Usability:** A website should be easy to navigate and understand. This means designing clear, consistent navigation menus, using straightforward language, and ensuring that all interactive elements are easy to find and use. User-friendly error messages and accessible design principles (such as ensuring the site is usable with screen readers) enhance the experience for all users, making the site more welcoming and efficient to use.
* **Reliability:** A reliable website is always available when users need it. This involves using reliable hosting services, implementing failover systems, and having **robust error handling mechanisms**. When issues do arise, the website should handle errors gracefully, providing clear messages to users and recovering quickly to minimize downtime. Regular monitoring and maintenance help ensure the site remains stable and responsive.
* **Maintainability:** well-designed website is easy to update and maintain. This is achieved through a modular design, where the site is broken down into **independent components**. Each component can be updated or replaced without affecting the entire site, making it easier to implement new features, fix bugs, and adapt to changing requirements. Clear documentation and coding standards also play a crucial role in maintaining the site's long-term health and scalability.

**7. Tech Stack**

**Frontend**

* **HTML5**: For structuring web content.
* **CSS3**: For styling web pages.
* **JavaScript**: For adding interactivity.

**Backend**

* **Node.js**: JavaScript runtime for server-side development.
* **Express**: A minimal and flexible Node.js web application framework.

**Database**

* **MySQL**: Relational database management system.
* **Sequelize**: ORM (Object-Relational Mapping) for MySQL and Node.js.

**Authentication**

* **JWT (JSON Web Tokens)**: For secure user authentication.
* **Express-Session**: Simple session management for Express applications.

**PDF Generation**

* **jsPDF**: For generating PDF files in the browser.

**Notification System**

* **Node-Cron**: For scheduling tasks like sending reminders.
* **Nodemailer**: For sending emails.

### 8. Other Requirements

* Regular backups of data
* Compliance with relevant data protection regulations (e.g., GDPR)

1.Introduction:

1.1 Purpose:

In the pursuit of enhancing operational efficiency and streamlining data management processes, we are developing a suite of three interrelated online platforms. These platforms are designed to automate the traditionally manual tasks associated with data entry, thus reducing errors, saving time, and improving overall productivity. The core functionalities of these platforms encompass login capabilities, dashboard management systems, request forms, and maintenance alert systems. Each of these components plays a crucial role in ensuring a seamless and integrated user experience, catering to various operational needs within an organization.

 **Login Capabilities**: The foundation of our online platforms is a secure and robust login system. This feature ensures that only authorized users can access the platforms, safeguarding sensitive information and maintaining data integrity. The login system will support various authentication methods, including single sign-on (SSO) and multi-factor authentication (MFA), to enhance security.

 **Dashboard Management Systems**: Central to the user experience is a comprehensive dashboard that provides users with a holistic view of relevant data and activities. The dashboard will offer customizable widgets and real-time analytics, enabling users to monitor key performance indicators, track progress, and make informed decisions based on up-to-date information.

 **Request Forms**: To facilitate efficient communication and workflow management, our platforms will include dynamic request forms. These forms will allow users to submit various types of requests, such as support tickets, resource allocations, and service inquiries. The forms will be designed to adapt to different needs, ensuring that all necessary information is captured accurately and promptly routed to the appropriate departments for action.

 **Maintenance Alert Systems**: Ensuring the smooth operation of any organization requires proactive maintenance and timely issue resolution. Our maintenance alert system will automatically notify relevant personnel of any maintenance needs or system issues, allowing for quick response and resolution. This system will help prevent downtime and maintain optimal performance of the organization's infrastructure.

By automating these key processes, our interrelated online platforms aim to enhance efficiency, accuracy, and responsiveness within the organization. This document will provide a detailed description of the requirements for each component, outlining the functionalities and features necessary to achieve these goals.

1.2 Scope:

Portal 1:

Portal 1 represents a pivotal tool in modernizing and optimizing the management of company requests within our organization. As a centralized platform, it offers a robust solution for users to submit a wide array of requests seamlessly while providing administrators with powerful tools to oversee and expedite the processing of these requests.

Portal 2:

Portal 2 is meticulously designed to adhere to the ISO 17025 standards, specifically tailored for managing machines and generating output templates and forms. This standard is critical for laboratories and testing facilities to ensure the reliability, accuracy, and consistency of their testing and calibration processes.

Portal 3:

Portal 3 is specifically designed to manage maintenance and expiry alerts for gas rooms within facilities, ensuring safety, compliance, and operational efficiency. Gas rooms typically house critical equipment or materials that require regular maintenance, monitoring, and adherence to safety protocols.

1.3 Definition, Acronyms and Abbreviation:

ISO:

ISO standards, or International Organization for Standardization standards, are internationally recognized frameworks and guidelines for ensuring products, services, and systems are safe, reliable, and of good quality. These standards are developed through a consensus-based process involving experts from various industries and countries. ISO standards cover a wide range of areas including quality management, environmental management, information security, risk management, and many others. They provide specifications for processes, products, and services, aiming to enhance compatibility, interoperability, and facilitate global trade. ISO standards are voluntary but widely adopted as benchmarks for best practices worldwide.

SRS:

Software Requirements Specification (SRS) is a detailed document that describes the functional and non-functional requirements of a software system. It serves as a blueprint for software development teams, outlining what the software should do and how it should perform.

PDF:

PDF stands for Portable Document Format. It is a file format developed by Adobe in the early 1990s to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems. PDF files can contain a variety of content, such as text, graphics, forms, annotations, and multimedia elements.

Node.js:

Node.js is a runtime environment that allows developers to run JavaScript code outside of a web browser, making it possible to build server-side applications and backend services using JavaScript. It utilizes an event-driven, non-blocking I/O model that makes it lightweight and efficient for handling concurrent requests and data-intensive operations.

HTML5:

HTML5, or Hypertext Markup Language version 5, is the latest evolution of the standard markup language used for creating and structuring web pages and web applications. It introduces new elements, attributes, and capabilities that enhance the functionality, multimedia support, and semantics of web content. HTML5 supports multimedia playback, offline web applications, responsive design, and improved accessibility, providing developers with powerful tools for building modern and interactive websites.

CSS3:

CSS3, or Cascading Style Sheets version 3, is the latest iteration of the CSS language used to style and format HTML documents. It introduces new features and enhancements that allow web designers and developers to create more visually appealing and responsive websites. CSS3 includes capabilities such as advanced selectors, transitions, animations, gradients, shadows, and flexible box layouts (Flexbox) and grid layouts (CSS Grid).

JavaScript:

JavaScript is a high-level, interpreted programming language primarily used to make web pages interactive and dynamic. It allows developers to add behavior to web pages, manipulate content, respond to user actions, and communicate asynchronously with servers. JavaScript is essential for front-end web development and increasingly used for server-side development with platforms like Node.js.

Express.js:

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications. It facilitates the creation of server-side applications and APIs (Application Programming Interfaces) by providing a streamlined and powerful set of tools and utilities.

Express-Session:

express-session is a middleware for Express.js, a Node.js web application framework. It enables session management by storing session data on the server and assigning a unique session identifier (session ID) to each visitor.

JWT:

JWT stands for JSON Web Token. It is a compact, URL-safe means of representing claims to be transferred between two parties. JWTs are typically used for authentication and information exchange in web applications and services.

MySQL:

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL). It is widely used for managing and storing structured data in databases. MySQL is known for its performance, reliability, and ease of use, making it a popular choice for web applications and large-scale projects. It supports various storage engines, transactions, replication, and offers comprehensive features for data manipulation, querying, and management. MySQL is compatible with many operating systems and integrates well with popular programming languages and frameworks, making it a versatile and widely adopted database solution.

Sequelize:

Sequelize is a promise-based Node.js ORM (Object-Relational Mapping) for SQL databases. It provides an easy-to-use interface for interacting with relational databases, abstracting away the complexities of SQL queries and database management. Sequelize supports multiple database dialects such as MySQL, PostgreSQL, SQLite, and MSSQL, allowing developers to work with different databases using the same API.

Nodemailer:

Nodemailer is a module for Node.js that simplifies the process of sending emails from a Node.js application. It supports sending emails with attachments, HTML content, and text content. Nodemailer integrates smoothly with various transport methods such as SMTP (Simple Mail Transfer Protocol), sendmail, and Amazon SES (Simple Email Service), among others.

Node-Cron:

Node-cron is a library for Node.js that allows you to schedule tasks (functions or scripts) to run at specified intervals, similar to cron jobs on Unix-like systems. It provides a simple API to define cron-like schedules using a syntax familiar to those who have worked with cron jobs. Node-cron handles the scheduling and execution of tasks in the background, making it useful for automating repetitive tasks such as data backups, report generation, or regular maintenance tasks in Node.js applications.

1.4 References:

<https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100424.pdf>

<https://www.npmjs.com/package/jspdf>

<https://www.mysql.com/>

<https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>

<https://expressjs.com/en/guide/routing.html>

<https://nodemailer.com/about/>

<https://www.npmjs.com/package/node-cron>

<https://www.cloudflare.com/en-in/learning/ssl/what-is-https/>

2. Overall Description:

2.1 Product Perspective:

In the pursuit of enhancing operational efficiency and streamlining data management processes, we are developing a suite of three interrelated online platforms. These platforms are designed to automate the traditionally manual tasks associated with data entry, thus reducing errors, saving time, and improving overall productivity. The core functionalities of these platforms encompass login capabilities, dashboard management systems, request forms, and maintenance alert systems. Each of these components plays a crucial role in ensuring a seamless and integrated user experience, catering to various operational needs within an organization.

 **Login Capabilities**: The foundation of our online platforms is a secure and robust login system. This feature ensures that only authorized users can access the platforms, safeguarding sensitive information and maintaining data integrity. The login system will support various authentication methods, including single sign-on (SSO) and multi-factor authentication (MFA), to enhance security.

 **Dashboard Management Systems**: Central to the user experience is a comprehensive dashboard that provides users with a holistic view of relevant data and activities. The dashboard will offer customizable widgets and real-time analytics, enabling users to monitor key performance indicators, track progress, and make informed decisions based on up-to-date information.

 **Request Forms**: To facilitate efficient communication and workflow management, our platforms will include dynamic request forms. These forms will allow users to submit various types of requests, such as support tickets, resource allocations, and service inquiries. The forms will be designed to adapt to different needs, ensuring that all necessary information is captured accurately and promptly routed to the appropriate departments for action.

 **Maintenance Alert Systems**: Ensuring the smooth operation of any organization requires proactive maintenance and timely issue resolution. Our maintenance alert system will automatically notify relevant personnel of any maintenance needs or system issues, allowing for quick response and resolution. This system will help prevent downtime and maintain optimal performance of the organization's infrastructure.

By automating these key processes, our interrelated online platforms aim to enhance efficiency, accuracy, and responsiveness within the organization. This document will provide a detailed description of the requirements for each component, outlining the functionalities and features necessary to achieve these goals.

2.2 Product Functions:

Portal 1: Company Request Portal and Dashboard

Portal 1 is designed to streamline and automate the handling of company requests. It serves as a centralized platform where users can submit various requests, and administrators can efficiently manage and track these requests through a dynamic dashboard.

1. **Request Submission Forms**:

**User-Friendly Interface**: The portal will feature intuitive and easy-to-navigate forms for request submissions. Users can submit requests for services, resources, support, and other needs.

**Dynamic Fields**: The forms will include dynamic fields that adjust based on the type of request, ensuring all necessary information is collected without overwhelming the user.

**File Attachments**: Users can attach relevant documents, images, or other files to their requests to provide additional context and information.

1. **Dashboard Management**:

**Centralized View**: The dashboard will provide a centralized view of all submitted requests, allowing administrators to quickly see the status, priority, and details of each request.

**Real-Time Updates**: Requests will be updated in real-time, reflecting any changes in status, new submissions, or completed actions instantly.

**Customizable Widgets**: The dashboard will offer customizable widgets that administrators can arrange according to their preferences, displaying key metrics, charts, and lists relevant to their roles.

1. **Request Tracking and Management**:

**Status Indicators**: Each request will have clear status indicators (e.g., pending, in progress, completed) to help users and administrators easily track the progress of their requests.

**Automated Notifications**: Automated email or SMS notifications will be sent to users and relevant staff members when the status of a request changes or when further action is needed.

**Search and Filter Options**: Advanced search and filter options will allow users and administrators to quickly find specific requests based on criteria such as date, type, priority, and status.

1. **User Roles and Permissions**:

**Role-Based Access Control**: Different user roles (e.g., requester, administrator, support staff) will have varying levels of access and capabilities within the portal, ensuring security and appropriate handling of requests.

**Permission Settings**: Administrators can set permissions to control who can view, edit, or manage specific types of requests and data within the system.

By incorporating these features, Portal 1 will significantly improve the efficiency and transparency of handling company requests, ensuring that all requests are processed promptly and accurately.

Portal 2: ISO-17025 Compliance with Template and form

Portal 2 is meticulously designed to adhere to the ISO 17025 standards, specifically tailored for managing machines and generating output templates and forms. This standard is critical for laboratories and testing facilities to ensure the reliability, accuracy, and consistency of their testing and calibration processes.

**Compliance with ISO 17025 Standards**:

Portal 2 is built from the ground up to comply with ISO 17025 guidelines, which set forth rigorous requirements for the competence of testing and calibration laboratories. These standards encompass various aspects, including management requirements, technical requirements, and quality assurance measures. By adhering to ISO 17025, Portal 2 ensures that all processes related to machine management and output template generation meet international benchmarks for quality and reliability.

**Machine Management**:

Central to Portal 2 is its comprehensive system for managing machines used in testing and calibration activities. The portal maintains detailed records of each machine’s specifications, calibration history, maintenance schedules, and usage logs. This information is crucial for ensuring that machines are operating within specified parameters and are capable of delivering accurate and consistent results as required by ISO 17025.

**Output Template and Form Generation**:

Portal 2 facilitates the creation and standardization of output templates and forms used in reporting testing and calibration results. Templates are designed to meet the specific requirements of ISO 17025, ensuring that reports are structured, consistent, and compliant with international standards. Users can generate customized forms and reports efficiently, incorporating relevant data fields, test parameters, measurement uncertainties, and other critical information as mandated by ISO 17025.

**Quality Assurance and Documentation**:

Portal 2 integrates robust quality assurance measures to uphold the integrity and reliability of testing and calibration processes. It includes mechanisms for verifying data accuracy, conducting internal audits, and managing corrective and preventive actions (CAPAs) to address any deviations from established procedures. Documentation within the portal is meticulously organized and maintained, facilitating traceability and transparency in compliance with ISO 17025 documentation requirements.

**Audit Trails and Traceability**:

To support regulatory compliance and accreditation processes, Portal 2 maintains detailed audit trails and ensures traceability of all activities related to machine management and output generation. Audit trails capture user actions, modifications to templates or forms, and updates to machine records, providing a clear historical record for auditing purposes. This feature helps laboratories demonstrate adherence to ISO 17025 standards during external audits and assessments.

**Training and Competence Management**:

Portal 2 includes features for managing training records and assessing the competence of personnel involved in testing and calibration activities. It tracks training certifications, proficiency testing results, and competency assessments to ensure that personnel are qualified to perform their assigned tasks in accordance with ISO 17025 requirements.

In essence, Portal 2 not only facilitates compliance with ISO 17025 standards but also enhances operational efficiency and quality within laboratories and testing facilities. By integrating comprehensive machine management capabilities with robust output template generation and adherence to stringent quality assurance practices, Portal 2 supports organizations in delivering reliable and accurate testing and calibration services while maintaining regulatory compliance and achieving accreditation objectives.

Portal 3: Maintenance and Expiry Alert for Gas room

Portal 3 is specifically designed to manage maintenance and expiry alerts for gas rooms within facilities, ensuring safety, compliance, and operational efficiency. Gas rooms typically house critical equipment or materials that require regular maintenance, monitoring, and adherence to safety protocols.

**Maintenance Management**:

Portal 3 provides a centralized platform for scheduling, tracking, and managing maintenance activities related to gas rooms. It allows facility managers and maintenance personnel to create maintenance schedules based on manufacturer recommendations, regulatory requirements, and internal policies. Scheduled maintenance tasks, such as inspections, calibration, and equipment servicing, can be assigned, monitored, and updated within the portal.

**Alerts and Notifications**:

The portal includes robust alert and notification systems to proactively manage maintenance activities and ensure timely interventions. Users can set up automated alerts for upcoming maintenance tasks, calibration deadlines, and equipment inspections. Notifications can be sent via email, SMS, or within the portal itself, alerting responsible personnel about pending tasks or approaching expiry dates for gas room-related activities.

**Compliance Tracking**:

Portal 3 facilitates compliance with regulatory standards and internal protocols governing gas room operations. It maintains comprehensive records of maintenance activities, calibration certificates, inspection reports, and safety audits. This documentation helps organizations demonstrate adherence to safety regulations and industry standards during inspections and audits.

**Expiry Alerts**:

In addition to maintenance alerts, Portal 3 monitors expiry dates for gas-related certifications, licenses, permits, and safety assessments. It automatically notifies administrators and responsible personnel well in advance of upcoming expirations, ensuring timely renewals and preventing operational disruptions due to lapsed certifications.

**Security and Access Control**:

Portal 3 incorporates robust security measures to safeguard sensitive information related to gas room maintenance and safety. Role-based access control ensures that only authorized personnel have access to specific functionalities and data within the portal. This helps prevent unauthorized modifications to maintenance schedules, alerts, and compliance records.

**Integration and Scalability**:

Portal 3 is designed for integration with existing facility management systems and scalable to accommodate future expansions or additional gas rooms within the organization. It supports interoperability with other software solutions for seamless data exchange and enhanced operational efficiency across the facility management landscape.

In summary, Portal 3 plays a critical role in ensuring the safe operation and regulatory compliance of gas rooms by facilitating proactive maintenance management, timely alerts for maintenance and expiry dates, and comprehensive tracking of compliance activities. By centralizing these functions within a single platform, organizations can enhance safety, streamline operations, and mitigate risks associated with gas room operations effectively.

2.3 User Classes and Characteristics;

Employees:

They are only allowed to view the data. They are not allowed to manipulate the data.

Maintenance Personnel:

They are allowed to do both the task i.e they can view the data and can also perform the manipulation of the data. They are ranked in the bottom most of the hierarchy.

Administrator:

He/She can also perform the same task as that of the maintenance personnel. He/She is ranked in the middle of the hierarchy.

Manager:

Manager is at the highest post in the hierarchy and he/she can do the both the task.

2.4 Operating Environment:

The system is designed to operate seamlessly on web browsers, making it highly accessible via the internet. This web-based approach ensures that users can access the platform from virtually anywhere, provided they have an internet connection. The decision to use web browsers as the primary interface brings several advantages in terms of accessibility, usability, and scalability.

**Cross-Platform Compatibility**:

By operating on web browsers, the system is inherently cross-platform, meaning it can be accessed from various devices and operating systems, including desktops, laptops, tablets, and smartphones. Whether users are running Windows, macOS, Linux, or mobile operating systems like Android and iOS, they can easily interact with the system without the need for specialized software installations. This compatibility ensures a broad reach and convenience for users, accommodating diverse working environments and personal preferences.

**Ease of Access and Use**:

Web-based systems are accessible through standard web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. This means users only need to enter a URL to access the platform, eliminating the complexities associated with downloading and installing dedicated applications. The intuitive nature of modern web browsers, combined with responsive design practices, ensures that the user interface adapts to different screen sizes and resolutions, providing an optimal user experience on any device.

**Scalability and Flexibility**:

Operating via web browsers allows the system to scale effortlessly to accommodate growing user bases and increased demand. Cloud-based hosting solutions can be leveraged to dynamically allocate resources, ensuring consistent performance even during peak usage periods. This scalability ensures that the system can grow with the organization and adapt to evolving needs without significant infrastructure changes.

**Security Considerations**:

While internet accessibility offers numerous benefits, it also necessitates robust security measures to protect sensitive data and ensure safe usage. The system will incorporate advanced security protocols, including HTTPS for secure data transmission, multi-factor authentication (MFA) for user verification, and regular security audits to identify and mitigate vulnerabilities. Access control mechanisms and encryption techniques will further safeguard data and maintain compliance with industry standards and regulations.

In summary, the system’s web-based operation via internet-accessible web browsers offers unparalleled convenience, accessibility, and scalability. It supports a wide range of devices and operating systems, enabling users to interact with the platform from any location. This approach not only enhances user experience and collaboration but also ensures that the system remains flexible, scalable, and secure, meeting the diverse needs of modern organizations.

2.5 Design and Implementation Constraints:

Design and implementation constraints for the system are significantly influenced by the need to comply with ISO 17025 standards, ensure a secure login mechanism, and maintain a responsive design. Each of these factors introduces specific requirements and challenges that must be carefully addressed to create a robust, reliable, and user-friendly platform.

**ISO 17025 Compliance**:

Adhering to ISO 17025 standards imposes stringent requirements on the system's design and implementation. This standard mandates rigorous procedures for the calibration and testing processes, ensuring accuracy, reliability, and traceability of results. Consequently, the system must incorporate features that support comprehensive documentation, detailed audit trails, and robust quality management processes. Implementing these requirements involves integrating functionalities for tracking equipment calibration, maintaining meticulous records, and facilitating internal and external audits. Additionally, the system must enable secure storage and retrieval of data, ensuring that all records are easily accessible and verifiable during compliance checks.

**Secure Login Mechanism**:

Ensuring a secure login mechanism is paramount to protect sensitive data and prevent unauthorized access. This constraint necessitates the implementation of advanced security protocols such as multi-factor authentication (MFA), strong password policies, and encryption of login credentials. The design must accommodate these security features seamlessly, ensuring that users can authenticate their identities without compromising on ease of use. Furthermore, the system must be capable of handling various authentication methods and integrating with external identity providers to support single sign-on (SSO) capabilities. This not only enhances security but also improves user convenience by reducing the number of login credentials users need to manage.

**Responsive Design**:

The requirement for a responsive design ensures that the system is accessible and usable across a wide range of devices and screen sizes. This involves creating a flexible and adaptive user interface that provides an optimal viewing experience, whether accessed on a desktop, tablet, or smartphone. Implementing a responsive design requires the use of modern web development frameworks and techniques such as fluid grids, flexible images, and media queries. The design must be tested across various devices and browsers to ensure consistency and functionality. Moreover, attention must be paid to performance optimization, ensuring that the system remains fast and responsive, even on devices with limited processing power and slower internet connections.

In conclusion, the constraints imposed by ISO 17025 compliance, secure login mechanisms, and responsive design collectively shape the design and implementation of the system. These factors necessitate a careful balance between stringent regulatory requirements, robust security measures, and a flexible, user-friendly interface. Addressing these constraints effectively will result in a system that not only meets high standards of accuracy and reliability but also provides a secure and accessible user experience across all devices.

PDF Generation:

Introduction:

PDF (Portable Document Format) is a widely used format for representing documents in a manner independent of application software, hardware, and operating systems. In the context of Node.js, PDFs are generated, manipulated, and managed using various libraries and tools. Node.js, being a server-side JavaScript runtime environment, offers several libraries to work with PDFs, making it an efficient choice for applications that need to generate or handle PDF documents

Benefits of PDFs in Manual Records:

 **Portability**: PDFs ensure that documents look the same on any device, which is crucial for maintaining consistency in manual records.

 **Security**: PDFs can be password-protected and encrypted, making them secure for storing sensitive information.

 **Compression**: PDFs can compress high-quality files to smaller sizes, which helps in saving storage space.

 **Integration**: PDFs can be integrated into various workflows, allowing easy sharing and collaboration.

 **Archiving**: PDFs are an ISO standard for archiving electronic documents, ensuring long-term preservation of records.

PDF Generation Technologies in Node.js:

 **PDFKit**: A popular library for generating PDFs with support for complex text layouts, vector graphics, and images.

 **Puppeteer**: A Node library which provides a high-level API to control Chrome or Chromium, often used for generating PDFs from HTML.

 **jsPDF**: A client-side JavaScript library that allows creation of PDF files directly within the browser.

 **PDFMake**: A library that offers declarative definition of PDFs with support for various document components and styling.

 **HummusJS**: A Node.js library that provides tools to create, modify, and parse PDF files.

Best PDF Generation Technology for Complex PDF Designs:

For complex PDF designs, **Puppeteer** is often considered the best choice. It allows developers to use HTML and CSS to design the layout of the PDF, leveraging the full power of web technologies to create intricate designs. This is particularly beneficial for those already familiar with web development.

As our requirements are limited hence, we are opting for jsPDF technology for the generation of pdfs. For complex design jsPDF pdf generation technology stands out when it comes to performance.

As we are only limited to generate the reports and receipt, the key feature of jsPDF is it can automatically generate the pdf once we have implemented it.

It requires less maintenance and it is easy to scale.

jsPDF:

jsPDF is a popular JavaScript library used to generate PDF documents in a web browser. It allows developers to create PDFs directly on the client-side, which can be downloaded or printed by users without requiring any server-side processing.

Features of jsPDF:

 **Text and Fonts**: Add text in different fonts and sizes.

 **Images**: Insert images (JPEG, PNG) into the PDF.

 **Shapes and Lines**: Draw shapes like rectangles, circles, and lines.

 **Tables**: Create tables with rows and columns.

 **Customization**: Style text, lines, and other elements with various options.

 **Plugins**: Extend functionality using plugins (e.g., jspdf-autotable for tables).

Common Use Cases:

 G**enerating Reports**: Create detailed reports with text, images, and tables.

 **Invoices and Receipts**: Automatically generate invoices or receipts in PDF format.

 **Exporting Data**: Allow users to export data from web applications into a PDF document.

 **Printable Forms**: Create and print forms that users can fill out by hand.

Here is an example, how we are going to implement jsPDF in our mainline production.



Conclusion:

By using jsPDF, you can create complex PDF documents right from the browser, making it a powerful tool for web applications that require PDF generation.

Notification System:

Introduction:

This presentation explores a notification system built with Node.js, utilizing two key libraries: Nodemailer and Node-Cron. Nodemailer streamlines email delivery within your application, handling connection, message creation, and sending logic. Node-Cron, on the other hand, empowers you to create scheduled tasks (cron jobs) that run periodically. With Node-Cron's familiar cron expression syntax, you can define precise delivery times for your notifications.

Nodemailer:

**Nodemailer** is a module for Node.js applications to easily send emails. It is designed to be simple and user-friendly, yet powerful enough to handle complex email requirements.

**Why Nodemailer is Used:**

1. **Email Communication**: To send transactional emails (like password resets, order confirmations) and marketing emails.
2. **Automation**: Automatically send emails triggered by events in the application.
3. **Testing**: Simulate email sending during development without actually sending emails using test services.

**Benefits of Nodemailer:**

1. **Ease of Use**: Simple API that allows for quick setup and integration.
2. **Compatibility**: Supports multiple email services including Gmail, Outlook, and SMTP.
3. **Customization**: Allows customization of email content, attachments, and HTML formatting.
4. **Security**: Supports secure email sending through OAuth2, SSL, and TLS.

Here's a basic example of sending an email with Nodemailer:

A screen shot of a computer code

Description automatically generated

Nodemailer will send the notification to the officer via an email, describing the task to perform.

Node-Cron:

Node-cron is a powerful library for scheduling tasks (cron jobs) in Node.js applications. It allows you to define tasks that run periodically based on the familiar cron expression syntax used in Unix-like systems for scheduling jobs.

Here are some key features of node-cron:

* **Cron Expression Support:** Node-cron lets you define schedules using cron expressions. These expressions specify the minute, hour, day of month, month, and day of week for when you want your task to run. This offers a flexible and expressive way to define complex scheduling patterns.
* **Simple API:** The library provides a straightforward API for creating and scheduling tasks. You can define a function to be executed and specify the cron expression for its execution schedule.
* **Multiple Schedules:** You can create multiple cron jobs within your application, each with its own schedule and task. This allows for versatile automation within your Node.js environment.

A computer screen shot of a program

Description automatically generated

While sending about notification from the inventory we have to fix some parameters that will automatically initialize the work of sending the mail to the respective officer or person in charge of the respective chamber.

Complete Backend Implementation:  
  
**Portal 1: Request Handling**

##### **3.1.1 Login Page:**

**Login Request:**

User submits login form with username and password. Backend receives the request, validates input, and retrieves the user from the database. If user is found and password matches (using bcrypt), a JWT token is generated and returned to the user.

**Password Recovery Request:**

User submits password recovery form with their email. Backend receives the request, validates input, and retrieves the user from the database. If user is found, a JWT reset token is generated and saved to the user's record. An email with the reset link containing the JWT token is sent to the user.  
Step: Create Password Reset Route

**Password Reset:**

User clicks on the reset link in the email, which leads to the reset form on the website. User submits the reset form with the new password and the reset token. Backend validates the token, checks its expiry, and if valid, updates the user's password (hashed) and clears the reset token fields.

Step: Configure Sequelize and Database  
 Creating Express Routes for Login and Password Recovery   
 Hashing Passwords Before Saving to Database

##### **3.1.2 Dashboard**

### Fetching User-Specific Data:

User accesses the dashboard. Backend verifies the JWT token to authenticate the user. Backend fetches user-specific pending requests and unread notifications from the database. Backend responds with the fetched data to be displayed on the dashboard.  
Step: Defining the Models and Relationships: Extend the existing models to include requests and notifications.  
 Creating Express Routes for Dashboard Data: Define routes to fetch user-specific data, request forms, and notifications.

**Providing Access to Request Forms:**

User navigates to the section for request forms. Backend verifies the JWT token to authenticate the user. Backend fetches and sends the list of available request forms to the user.  
Step: Creating a Route to Serve Request Forms.

**Handling Notifications and Alerts:**

User subscribes to real-time notifications using Socket.IO. Backend sends notifications to subscribed users in real-time whenever a new notification is created or an event occurs. Notifications are stored in the database and sent to the user via WebSocket.  
Step: Set up Socket.IO for Real-Time Notifications: Configure Socket.IO in application.

**3.1.3 Request from the Company - Landing Page + Form:**

**Form Submission:**

User fills out the request form on the landing page and submits it. Form data is sent to the backend via a POST request to /submit-request.  
Step: Defining the Models: Extend the existing models to include request details.

Creating Express Routes for Handling Requests: Define routes to handle request form submissions and input validation.

**Input Validation:**

Backend receives the request and validates the input data using express-validator. Validation checks ensure that the title is at least 5 characters long and the description is at least 10 characters long. If validation fails, an error response is sent back to the user with details of the validation errors.  
Step: Input Validation Using Express-Validator: Implement validation middleware in the route handling the request form submission.

**Storing Request Details:**

If validation passes, the backend verifies the JWT token to authenticate the user. The backend creates a new request record in the database, associating it with the authenticated user's ID. The new request is saved to the Request table in the database.

**Response:**

Backend responds with a success message and the details of the newly created request. If there's a server error, an appropriate error message is sent to the user.

**3.1.4 Receiving Page for the Request:**

**Fetching the List of Received Requests:**

Admin or authorized user accesses the receiving page. Backend verifies the JWT token to authenticate the user. Backend fetches the list of all received requests from the database. Backend responds with the fetched data to be displayed on the receiving page.

**Processing Approval or Rejection of Requests:**

Admin or authorized user selects a request to approve or reject. User sends a request to the backend with the request ID and action (approve or reject). Backend verifies the JWT token to authenticate the user. Backend finds the request by its ID and updates its status to either approved or rejected. Backend responds with a success message.  
Step: Creating Routes for Approval and Rejection: Define routes to handle approval and rejection of requests.

**Implementing Search and Filter Functionalities:**

Admin or authorized user uses search and filter options on the receiving page. Backend verifies the JWT token to authenticate the user. Backend processes the search and filter criteria from the query parameters. Backend fetches the filtered list of requests from the database based on the criteria. Backend responds with the filtered data to be displayed on the receiving page.

Step: Creating Routes for Searching and Filtering Requests:Define routes to search and filter received requests based on criteria like status and title.

#### 3.2 Portal 2: ISO-17025 Compliance

##### **3.2.1 Manual in Drop-down Format:**

**Providing Hierarchical Data Structure of the ISO-17025 Manual:**

The backend fetches manual sections from the database. Sections are organized hierarchically (parent and child relationships). Backend sends the hierarchical data structure to the frontend. Frontend displays the manual sections in a drop-down format based on the hierarchical structure.  
Step: Defining the Manual Section Model: To the structure of the manual sections and their relationships (parent-child).  
 Creating Express Routes for Fetching Manual Data: Implement routes to fetch manual sections in a hierarchical format and search manual content.

**Supporting Search Queries for Manual Content:**

User enters a search query for specific content in the manual. Backend processes the search query and retrieves relevant sections. Backend sends the search results to the frontend. Frontend displays the search results to the user, allowing them to navigate directly to relevant manual sections.  
  
**3.2.2 Employee Login Page:**  
  
**User Authentication:**

User submits username and password via the login form. Backend validates credentials against the database (User model). If credentials are valid, the backend generates a JWT token. JWT token is sent back to the client for subsequent authorized requests.  
Step: Defining the structure of the User model to store user credentials and roles.

Implementing User Authentication Routes: by creating routes to handle user login and issue JWT tokens upon successful authentication.

**Displaying User Hierarchy Based on Roles:**

After successful authentication, the frontend sends the JWT token in subsequent requests. Backend verifies the JWT token using middleware (authenticateJWT). Based on the user's role stored in the token (req.user.role), the backend fetches users with similar roles. The backend sends the filtered user hierarchy data to the frontend for display.  
Step: Implementing Authorization Middleware: By creating middleware to verify JWT tokens and authorize access based on user roles.

Route to Fetch User Hierarchy Based on Roles: Create a route to fetch and display user hierarchy based on roles.

**3.2.3 Landing Page (Services & Types of Test):**  
  
**Fetching and Displaying a List of Services:**

Frontend initiates a request to /services endpoint. Backend retrieves all services from the Service model in the database. Backend sends the list of services (Service objects) as a JSON response to the frontend. Frontend displays the list of services on the landing page.

**Fetching Detailed Information on Each Type of Test:**

User selects a specific service from the list displayed on the landing page. Frontend sends a request to /services/:id endpoint, where :id is the ID of the selected service. Backend fetches detailed information of the service by its ID (findByPk method). Backend sends the detailed service information as a JSON response to the frontend. Frontend displays the detailed information of the selected service, including its description.  
  
**3.2.4 Template of the Documents**

**Providing Download Links for Document Templates:**

Users access a designated endpoint (e.g., /templates/:id) to download specific templates identified by :id.

Backend verifies permissions and serves the corresponding document file for download.

**Storing and Retrieving Document Templates:**

Document templates are uploaded and securely stored in a designated folder or database table.

Backend provides endpoints for uploading new templates and querying existing templates.

#### 3.2.5 Create Simple Forms

**Saving Created Forms and Their Data to the Database:**

##### User configurations for forms (fields, validations) are stored in a structured format (e.g., JSON) in the database.

Form submissions are saved securely, associating each submission with its respective form configuration.

**Validating Form Data:**

Backend verifies form submissions against predefined rules and validations (e.g., required fields, data formats).  
Error messages are generated for invalid submissions, guiding users to correct their input.

**Providing Template Selection Options:**

Users choose from available form templates or create new forms from scratch.

Backend offers a user-friendly interface to manage and select templates based on user preferences.

#### 3.2.6 Export to PDF

**Generating PDF Files from Form Data:**

Backend retrieves form submissions from the database and converts relevant data into PDF format.

Formatting and styling are applied to ensure PDFs are consistent and professional.

**Handling Requests to Download or Email the Generated PDFs:**

Users initiate requests to download PDFs via specific endpoints (e.g., /download/:id). Backend verifies permissions and serves the requested PDF file for download. Optionally, users may request PDFs to be sent via email, leveraging backend functionality to handle email delivery securely.

### Portal 3: Entries & Maintenance

#### 3.3.1 Gas Room Expiry & Maintenance Levels

**Data Entry for Gas Maintenance Schedules:**

Users access a form or interface to input maintenance schedules for each gas room. Backend validates and stores the entered data securely in a database, associating each entry with the corresponding gas room.

**Alert System for Nearing Expiry/Maintenance Dates:**

Backend regularly checks the database for upcoming maintenance or expiry dates based on configured intervals. When nearing dates are detected, the backend triggers alerts through multiple channels (e.g., pop-ups in the application, email notifications). Alerts include relevant details (e.g., gas room ID, type of maintenance, upcoming date) to facilitate timely actions by users.

#### 3.3.2 Reminding via Pop-up/Alert

**Pop-up Alerts for Maintenance/Expiry:**

Implement real-time pop-up alerts within the web application to notify users immediately when maintenance or expiry limits are reached. Alerts should be displayed prominently to ensure they capture user attention effectively.

#### External Interface Requirements

#### 4.1 User Interfaces

The system will provide various user interfaces to facilitate interaction with different functionalities. Each interface is designed to ensure usability, accessibility, and efficiency for users across different roles within the organization.

##### **4.1.1 Web Pages for User Interfaces**

**Login Page**:

* + **Description**: Secure login page for users to authenticate and access the system.
  + **Functional Requirements**:
    - User authentication (username/password).
    - Password recovery option (via email).
    - Integration with authentication services (e.g., JWT for token-based authentication).
    - Error handling for invalid login attempts.

**Dashboards**:

* + **Description**: Comprehensive dashboard displaying relevant data and key performance indicators (KPIs).
  + **Functional Requirements**:
    - Customizable widgets and real-time analytics.
    - Overview of pending requests, notifications, and alerts.
    - Interactive charts and graphs for data visualization.
    - Responsive design for access from various devices.

**Forms**:

* + **Description**: Dynamic forms for submitting requests and capturing data.
  + **Functional Requirements**:
    - Form builder with drag-and-drop interface.
    - Validation of form inputs to ensure data integrity.
    - Storage of form submissions in the database.
    - Template selection for creating new forms.

**Templates**:

* + **Description**: Repository of document templates available for download.
  + **Functional Requirements**:
    - Provision of download links for document templates.
    - Storage and retrieval of templates from a centralized database.
    - Version control and access permissions for templates.

**Alerts**:

* + **Description**: System alerts for notifying users about critical events and updates.
  + **Functional Requirements**:
    - Real-time pop-up alerts within the application interface.
    - Email notifications for urgent alerts.
    - Configuration options for alert thresholds and recipients.

##### **4.1.2 Software Interfaces**

* **Database Integration**:
  + Integration with MySQL database using Sequelize ORM for storing and retrieving application data, including user information, form submissions, templates, and maintenance schedules.
* **jsPDF Library**:
  + Integration with jsPDF library for generating PDF documents dynamically from form submissions and other application data.

##### **4.1.3 Communication Interfaces**

* **HTTPS Protocol**:
  + Secure data transmission over the web using HTTPS to protect user data during communication between client devices and the server.

#### Backend Technologies:

**Node.js**: Server-side JavaScript runtime environment for building scalable and fast network applications.

**Express.js**: Minimalist web application framework for Node.js, providing robust features for building APIs and web applications.

**MySQL / PostgreSQL / MongoDB**: Relational or NoSQL databases for storing structured data (MySQL with Sequelize ORM for relational databases, MongoDB for NoSQL databases).

**Sequelize / Mongoose**: Object-Relational Mapping (ORM) libraries for Node.js, facilitating interaction with databases and managing data models.

**JSON Web Tokens (JWT)**: Authentication mechanism for securely transmitting information between parties as JSON objects, used for user authentication and authorization.