PROJECT TITLE

DGT SENSOR LIFE EXPIRY TRACKING AUTOMATIC MECHANISM TO BE DEVELOPED

RELIANCE
JAMNAGAR

DTA-DGT SENSOR TRACKING WEB APP

REPORT

PRESENTED BY:- MR. AMIT RAJ
EC :- 19603903
MENTOR :- MR. BHAVIK VAGHAMSHI

Table of CONTENTS

01 **Introduction**
- Overview of the Streamlit App
- Purpose and Goals of the App
- Technologies Used

02 **App Architecture**
- High-Level Architecture
- Component Breakdown

03 **Setup and Installation**
- Dependencies and Requirements
- Installation Instructions

04 **User Interface (UI)**
- Layout and Design
- Key UI Elements
- User Interaction

05 **Core Features and Functionalities**
- Login and Authentication
- Password Management
- Selection Page
- Main Application Features
  - -Uploading and Filtering Data
  - -Data Visualization
  - - Data Export Options

06 **Backend Logic and Data Processing**
- Data Input and Handling
- Data Processing Workflows
- Integration with External Services

Table of CONTENTS

07 **Code Analysis**
- Code Structure and Organization
- Key Functions and Methods
- Code Efficiency and Optimization

08 **Data Visualization**
- Types of Visualizations Used
- Implementation Details
- Interaction and Customization

09 **Error Handling and Debugging**
- Common Errors and Solutions
- Debugging Techniques

10 **Performance Considerations**
- Performance Metrics
- Optimization Strategies

11 **Security Aspects**
- Security Measures Implemented
- Best Practices for Secure Coding

12 **Testing and Validation**
- Types of Tests Conducted
- Testing Frameworks and Tools Used
- Validation Results

13 **Deployment**
- Deployment Steps
- Hosting Options
- Continuous Integration/Continuous Deployment (CI/CD) Setup

Table of CONTENTS

14 **Documentation and User Support**
- Documentation Provided
- User Support Mechanisms

15 **Conclusion**
- Summary of Finding
- Future Enhancements
- Final Remarks

16 **WorkFlow Diagrams**
- Login Page flow diagram
- Main Page flow diagram

17 **UI & Features commented Snapshot**
- Login Page
- Selection Page
- Main Page

18 **Appendix**
- Raw Code
- Github Repo :- https://github.com/Amit-Raj26/DTA-DGT_SensorTrack/

19 **Other Experiences at RIL**
- Centeral Engineering Servce Hands-on Workshop
- Aromatics Plant Visit
- Centeral Engineering Servce Plant field Visit
  - -Day1 :- Field Instrumentation calibration & Troubleshooting Lab
  - -Day2 :- Valve Maintainence Shop

1. Introduction

Overview of the Streamlit App

The Streamlit app, named DTA-DGT_SensorTrack, is designed to facilitate the tracking of sensor procurement and replacement activities within an organization. The app allows users to upload Excel files containing sensor data, filter the data based on various criteria, and download the filtered data in either Excel or PDF format. This tool enhances the efficiency of managing sensor-related activities by providing an intuitive and interactive user interface.

Purpose and Goals of the App

The primary purpose of the app is to streamline the processes of sensor procurement and replacement. The goals include:

- Providing a user-friendly interface for data upload and visualization.
- Enabling detailed filtering of sensor data based on multiple criteria.
- Allowing users to export filtered data in preferred formats for further analysis or reporting.

Technologies Used

- Streamlit: A framework for building interactive web applications in Python.
- Pandas: A powerful data manipulation and analysis library in Python.
- PDFKit: A library for converting HTML content to PDF documents.
- ExcelWriter: A pandas utility for writing data to Excel files.
- WKHTMLTOPDF: A command-line tool to render HTML into PDF, used via PDFKit.

2. App Architecture

High-Level Architecture

The app is structured into several key components, each responsible for different functionalities:

- Login System: Manages user authentication.
- Password Management: Allows users to change or reset their passwords.
- Selection Page: Enables users to choose between tracking sensor procurement or replacement.
- Main Application: Handles data upload, filtering, visualization, and export.

3. Setup and Installation

Dependencies and Requirements

The application requires the following dependencies:

- streamlit
- pandas
- pdfkit
- xlsxwriter
- wkhtmltopdf (for PDF generation)

Installation Instructions

To install the required dependencies, execute the following commands:

```sh
pip install streamlit pandas pdfkit xlsxwriter
```

Ensure that wkhtmltopdf is installed and accessible in your system. On Windows, it can be downloaded from here

4. User Interface (UI)

Layout and Design

The UI of the app is designed for ease of use and clarity, featuring:

- A clean **login form** with fields for username and password.
- A **selection page** with buttons to choose between sensor procurement and replacement tracking.
- A comprehensive **main application page** with sections for data upload, filtering, and export options.

Key UI Elements

- **Forms** for login, changing passwords, and resetting passwords.
- **Buttons** for navigation and user actions like uploading files, filtering data, and downloading reports.
- **Sidebar** for filtering options and export settings.

User Interaction

The app provides an interactive user experience with elements like checkboxes, dropdowns, and buttons that dynamically update the UI based on user actions.

5. Core Features & Functionalities

Login and Authentication

The login system ensures that only authorized users can access the app. It uses session state to manage login status and redirects users to the appropriate page based on their authentication status.

Password Management

- Change Password: Users can change their password by providing the old and new passwords.
- Forgot Password: Users can reset their password by entering their phone number and OTP.

Selection Page

- Users can choose between tracking sensor procurement and replacement activities.

Main Application Features

- Uploading and Filtering Data: Users can upload Excel files containing sensor data and apply filters based on plant, make, model, gas type, and date range.
- Data Visualization: Filtered data is displayed in a table format for easy viewing.
- Data Export Options: Users can export the filtered data in Excel or PDF format.

6. Backend Logic and Data Processing

Data Input and Handling

The app accepts Excel files and processes them using pandas. It ensures that date fields are correctly converted to datetime objects for accurate filtering.

Data Processing Workflows

Filtering: Data is filtered based on user-selected criteria such as plant, make, model, gas type, and date range.
Conditional Logic: The app dynamically adjusts filtering criteria based on user selections (e.g., procurement vs. replacement).

Integration with External Services

PDFKit: Used for generating PDF reports from HTML content.
ExcelWriter: Used for generating Excel reports from pandas DataFrames.

7. Code Analysis

Code Structure and Organization

The code that is mentioned in appendix of this report is modular, with functions defined for each major feature:

- send_otp: Generates and displays OTP for password reset.
- login: Renders the login form and handles user authentication.
- change_password: Provides a form for changing the user password.
- forgot_password: Handles password reset via OTP.
- selection_page: Displays options for sensor tracking.
- main: Handles the main application logic including data upload, filtering, and export.

Key Functions and Methods

- Login Functions: login, change_password, forgot_password
- Data Handling Functions: convert_df_to_excel, convert_df_to_pdf
- UI Rendering Functions: selection_page, main

Code Efficiency and Optimization

The app uses pandas for efficient data manipulation and ensures responsive UI updates using Streamlit's rerun and session state features.

8. Data Visualization

Types of Visualizations Used

- Data Tables: Display filtered data in a tabular format.
- Downloadable Reports: Filtered data can be exported to Excel and PDF formats

Implementation Details

- Pandas DataFrame: Used for data manipulation and display.
- Streamlit Widgets: Used for interactive elements like file uploader, dropdowns, and buttons.

Interaction and Customization

- Dynamic Filtering: Users can interactively filter data based on multiple criteria.
- Custom Export Options: Users can customize the columns to be included in the export

9. Error Handling and Debugging

Common Errors and Solutions

- Invalid Credentials: Displays an error message if login credentials are incorrect.
- OTP Mismatch: Displays an error message if the entered OTP does not match the generated OTP

Debugging Techniques

- Logging: Use of st.warning and st.error for displaying error messages.
- Session State Management: Use of st.session_state for managing app state and debugging issues related to state changes

10. Performance Considerations

Performance Metrics

- Data Processing Time: Efficient handling of large datasets using pandas.
- UI Responsiveness: Ensures that the UI remains responsive even with large datasets.

Optimization Strategies

- Efficient Data Manipulation: Uses pandas for optimized data manipulation.
- Asynchronous Processing: Consider using asynchronous processing for long-running tasks.

11. Security Aspects

Security Measures Implemented

- Session Management: Uses st.session_state for secure session management.
- Password Handling: Provides functionality for secure password change and reset.

Best Practices for Secure Coding

- Secure OTP Handling: Ensure that OTPs are sent securely in a real application.
- Input Validation: Validate all user inputs to prevent security vulnerabilities.

12. Testing and Validation

Types of Tests Conducted

- Unit Testing: Test individual functions and methods for expected behavior.
- Integration Testing: Test the integration of different components.

Testing Frameworks and Tools Used

- Streamlit's Testing Tools: Utilize Streamlit's built-in tools for testing the app.

Validation Results

- Functionality Tests: Ensure that all features work as expected.
- User Experience Tests: Ensure that the UI is intuitive and user-friendly

13. Deployment

Deployment Steps

1. Setup Environment: Install necessary dependencies and set up the environment.
2. Deploy to Server: Use a hosting service to deploy the app.

Hosting Options

- Streamlit Sharing: Deploy the app using Streamlit's hosting service. The app is currently deployed on Streamlit Community Cloud itself.
- Custom Server: Deploy the app on a custom server using Docker or similar tools using Github link provided in this report

Continuous Integration/Continuous Deployment (CI/CD) Setup

- CI/CD Tools: Use tools like GitHub Actions or Jenkins for automated testing and deployment.

14. Documentation and User Support

Documentation Provided

- User Guide: Refer to the Github link provided in the appendix to find the user guide
- Code Documentation: Include comments and documentation within the code.

User Support Mechanisms

- Help Section: Include a help section within the app for user assistance.
- Contact Information: +91-7464055998

15. Conclusion

Summary of Findings

- The Streamlit app provides a robust solution for tracking sensor procurement and replacement activities.
- The app is user-friendly, with interactive features and options for data export.

Future Enhancements

- Enhanced Security: Implement secure OTP handling and input validation.
- Additional Features: Add more features for data visualization and reporting.

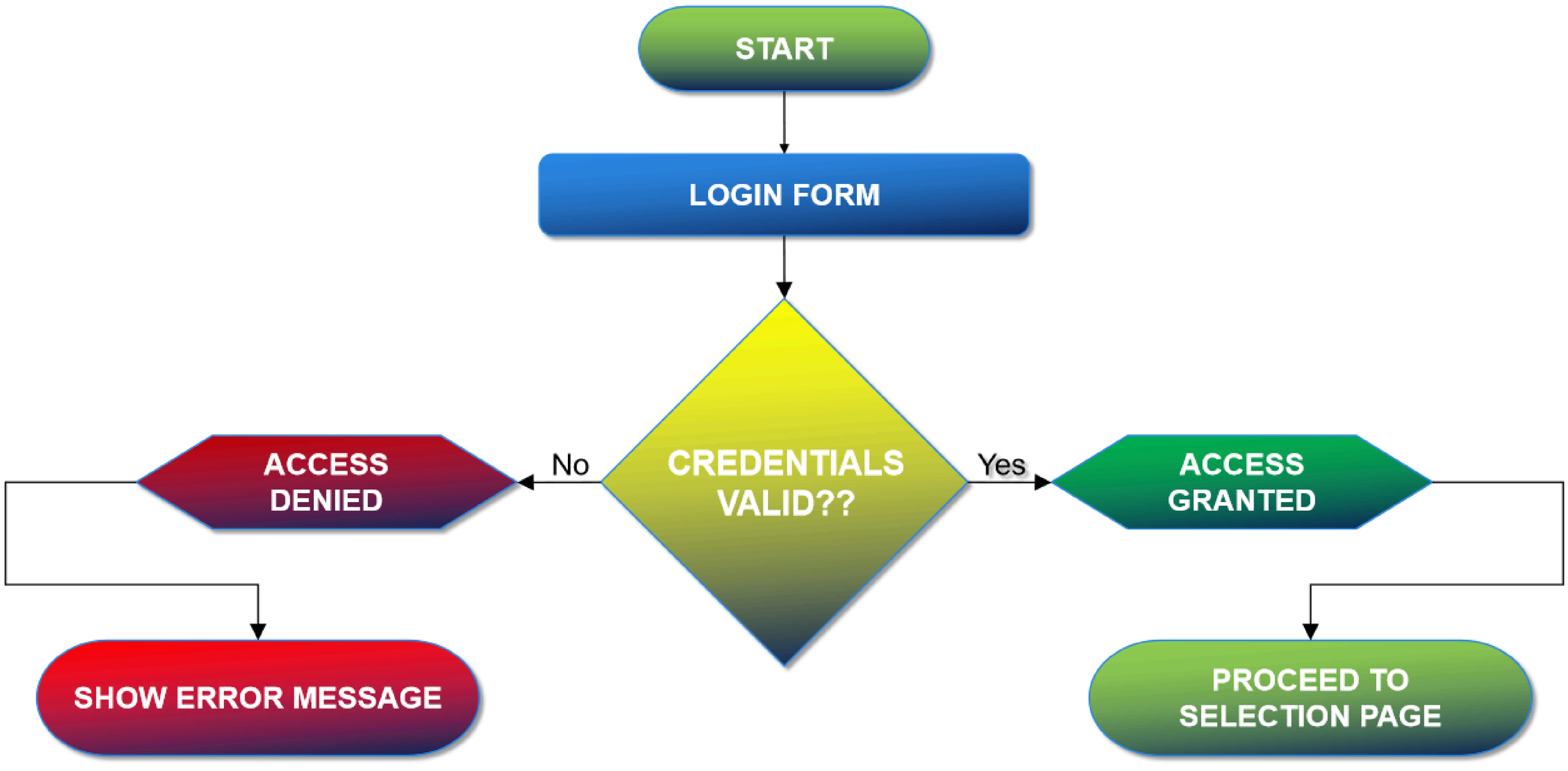
Final Remarks

- The app successfully meets its goals of facilitating sensor tracking and data management

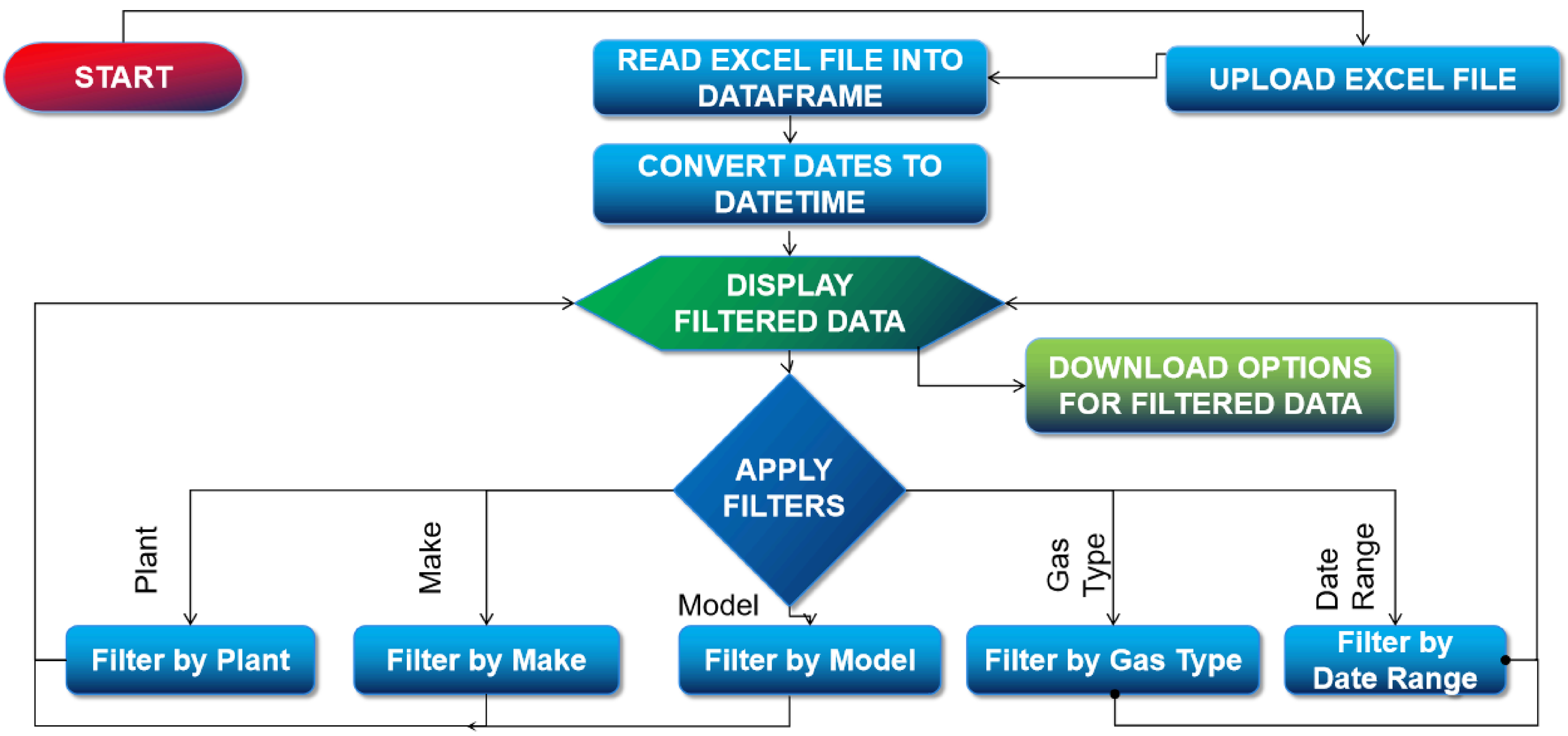
PTO
(Please turn over)

16. WorkFlow Diagrams

LOGIN PAGE WORKFLOW DIAGRAM



MAIN PAGE WORKFLOW DIAGRAM

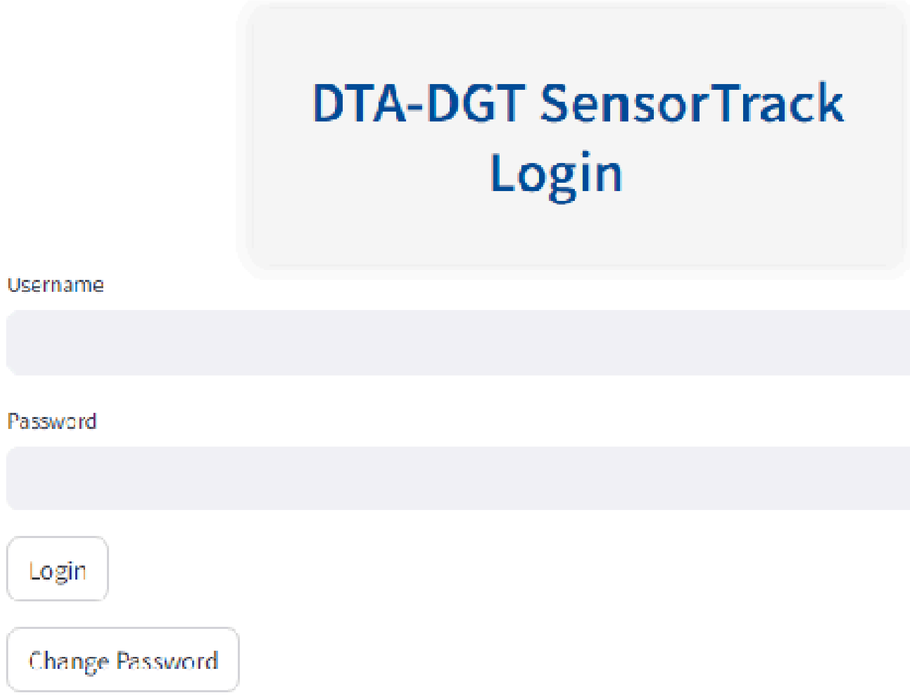


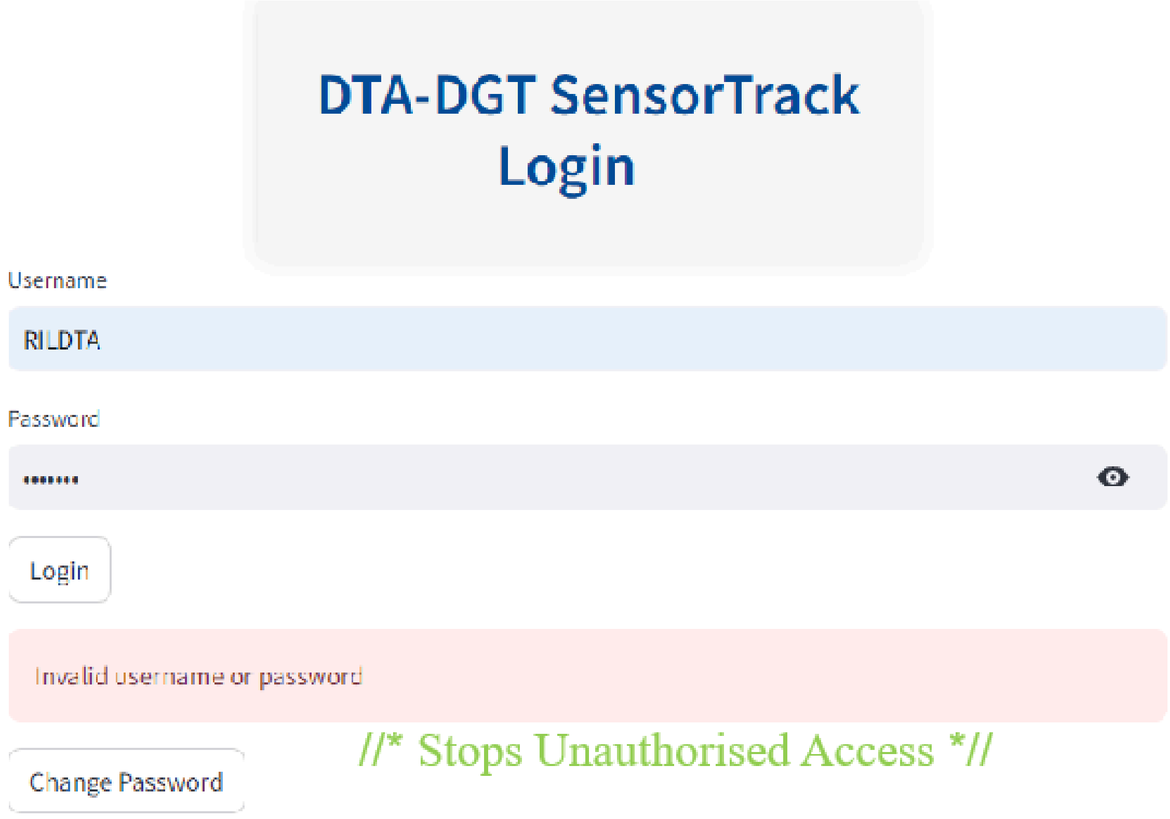
17. UI & Features commented Snapshot

Login Page of DTA_DGT-SensorTrack

Paste this link to your browser to access the app :-

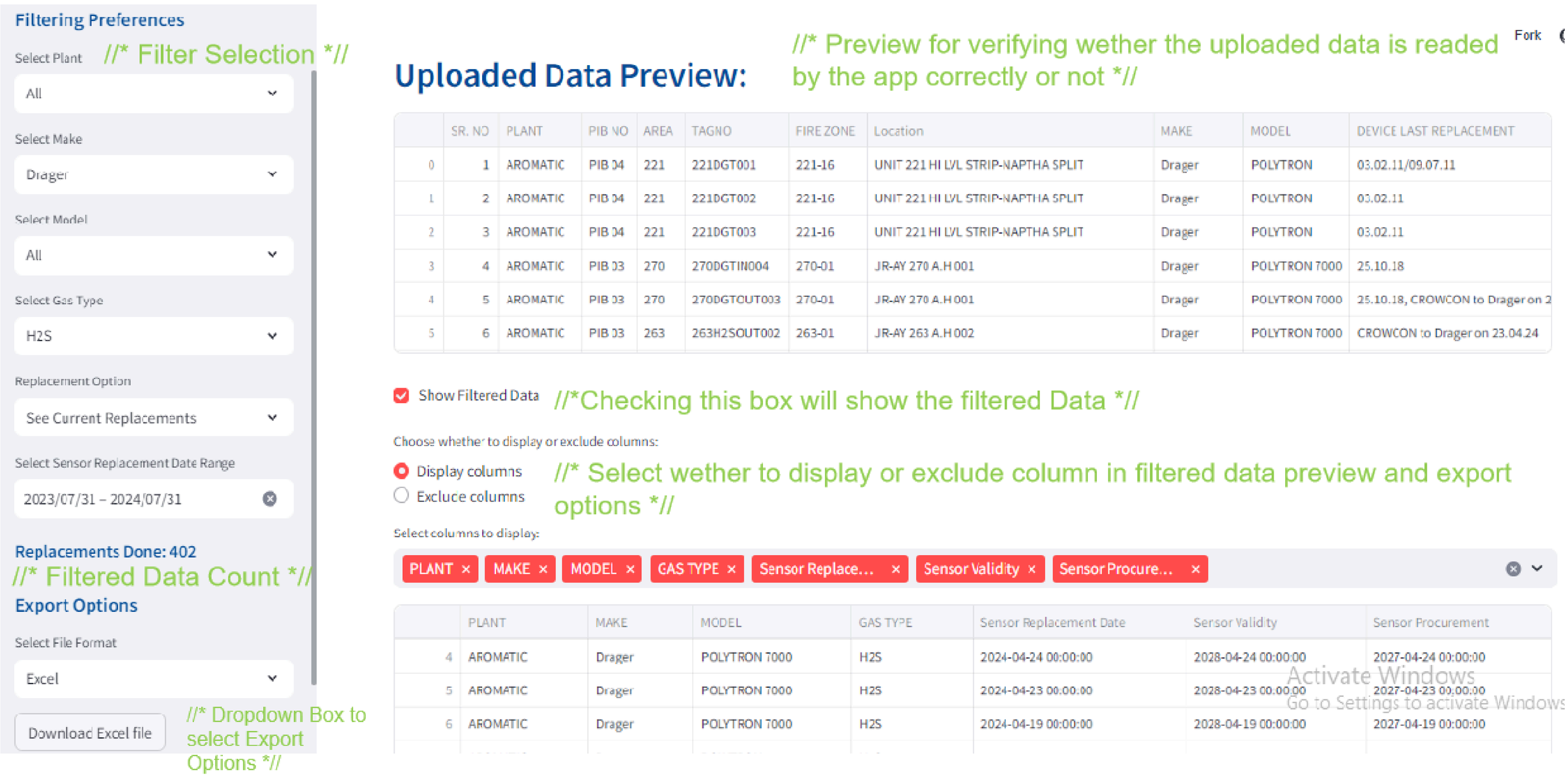
Github Link to the app :- https://github.com/Amit-Raj26/DTA-DGT_SensorTrack/tree/main





//* Stops Unauthorised Access *//

Main Page of DTA_DGT-SensorTrack



	SR. NO	PLANT	PIB NO	AREA	TAGNO	FIRE ZONE	Location	MAKE	MODEL	DEVICE LAST REPLACEMENT
0	1	AROMATIC	PIB 04	221	221DGT001	221-16	UNIT 221 HI L/L STRIP-NAPTHA SPLIT	Drager	POLYTRON	03.02.11/09.07.11
1	2	AROMATIC	PIB 04	221	221DGT002	221-16	UNIT 221 HI L/L STRIP-NAPTHA SPLIT	Drager	POLYTRON	03.02.11
2	3	AROMATIC	PIB 04	221	221DGT003	221-16	UNIT 221 HI L/L STRIP-NAPTHA SPLIT	Drager	POLYTRON	03.02.11
3	4	AROMATIC	PIB 03	270	270DGTIN004	270-01	JR-AY 270 A.H 001	Drager	POLYTRON 7000	25.10.18
4	5	AROMATIC	PIB 03	270	270DGTOUT003	270-01	JR-AY 270 A.H 001	Drager	POLYTRON 7000	25.10.18, CROWCON to Drager on 2
5	6	AROMATIC	PIB 03	263	263H2SOUT002	263-01	JR-AY 263 A.H 002	Drager	POLYTRON 7000	CROWCON to Drager on 23.04.24

	PLANT	MAKE	MODEL	GAS TYPE	Sensor Replacement Date	Sensor Validity	Sensor Procurement
4	AROMATIC	Drager	POLYTRON 7000	H2S	2024-04-24 00:00:00	2028-04-24 00:00:00	2027-04-24 00:00:00
5	AROMATIC	Drager	POLYTRON 7000	H2S	2024-04-23 00:00:00	2028-04-23 00:00:00	2027-04-23 00:00:00
6	AROMATIC	Drager	POLYTRON 7000	H2S	2024-04-19 00:00:00	2028-04-19 00:00:00	2027-04-19 00:00:00

//* Filter Selection *//

//* Filtered Data Count *//

//* Dropdown Box to select Export Options *//

//* Preview for verifying wether the uploaded data is readed by the app correctly or not *//

//*Checking this box will show the filtered Data *//

//* Select wether to display or exclude column in filtered data preview and export options *//

18. Appendix

Raw Code

DTA_DGT SensorTrack

Code

```
import streamlit as st
import pandas as pd
from datetime import datetime
from io import BytesIO
import random
import subprocess
import os
# Ensure wkhtmltopdf is installed
if not os.path.exists('setup.sh'):
    st.warning("wkhtmltopdf is not installed. Running setup script...")
    subprocess.run(['bash', 'setup.sh'], check=True)
import pdfkit

st.set_page_config(page_title="DTA-DGT_SensorTrack", layout="wide")

# Placeholder for the OTP (in a real application, this should be handled more securely)
otp_placeholder = st.empty()

# Initial username and password
initial_username = 'RILDTA'
initial_password = 'Rilfng@1234'

# Session state initialization
if 'logged_in' not in st.session_state:
    st.session_state.logged_in = False
if 'password' not in st.session_state:
    st.session_state.password = initial_password
if 'username' not in st.session_state:
    st.session_state.username = initial_username
if 'otp' not in st.session_state:
    st.session_state.otp = ''
if 'page' not in st.session_state:
    st.session_state.page = 'login'
if 'track_option' not in st.session_state:
    st.session_state.track_option = 'sensor_procurement'

# Function to send OTP
def send_otp():
    st.session_state.otp = str(random.randint(100000, 999999))
    otp_placeholder.text(f"Your OTP is: {st.session_state.otp}")  # This simulates sending the OTP

# Function to render the login form
def login():
    st.markdown("""
    <style>
    .login-form {
        max-width: 400px;
        margin: auto;
        padding: 20px;
        background-color: #f9f9f9;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .login-form h2 {
        text-align: center;
        color: #004d99;
```

```
        }
        .login-form input {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
        }
        .login-form button {
            width: 100%;
            padding: 10px;
            background-color: #004d99;
            border: none;
            color: white;
            border-radius: 5px;
            cursor: pointer;
        }
        .login-form button:hover {
            background-color: #003366;
        }
        .login-form .links {
            text-align: center;
            margin-top: 10px;
        }
        .login-form .links a {
            color: #004d99;
            text-decoration: none;
            margin: 0 10px;
        }
        .login-form .links a:hover {
            text-decoration: underline;
        }
        </style>
        <div class="login-form">
            <h2>DTA-DGT SensorTrack Login</h2>
        """, unsafe_allow_html=True)

        username = st.text_input("Username", key="login_username")
        password = st.text_input("Password", type="password", key="login_password")
        if st.button("Login", key="login_button"):
            if username == st.session_state.username and password == st.session_state.password:
                st.session_state.logged_in = True
                st.session_state.page = 'selection'
                st.success("Logged in successfully")
                st.rerun()  # Ensure the app reruns to reflect the state change
            else:
                st.error("Invalid username or password")

        # Hidden buttons to handle page changes
        if st.button("Change Password", key="change-password-link", on_click=lambda: change_page('change_password'))
            st.experimental_rerun()

        if st.button("Forgot Password", key="forgot-password-link", on_click=lambda: change_page('forgot_password'))
            st.experimental_rerun()

    # Function to change the page
    def change_page(page):
        st.session_state.page = page

    # Function to render the change password form
    def change_password():
        st.markdown("""
```

```
        <style>
        .change-password-form {
            max-width: 400px;
            margin: auto;
            padding: 20px;
            background-color: #f9f9f9;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .change-password-form h2 {
            text-align: center;
            color: #004d99;
        }
        .change-password-form input {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
        }
        .change-password-form button {
            width: 100%;
            padding: 10px;
            background-color: #004d99;
            border: none;
            color: white;
            border-radius: 5px;
            cursor: pointer;
        }
        .change-password-form button:hover {
            background-color: #003366;
        }
        </style>
        <div class="change-password-form">
            <h2>Change Password</h2>
        """, unsafe_allow_html=True)

        old_password = st.text_input("Old Password", type="password", key="change_old_password")
        new_password = st.text_input("New Password", type="password", key="change_new_password")
        confirm_password = st.text_input("Confirm New Password", type="password", key="change_confirm_password")
        if st.button("Change Password", key="change_password_button"):
            if old_password == st.session_state.password:
                if new_password == confirm_password:
                    st.session_state.password = new_password
                    st.success("Password changed successfully")
                else:
                    st.error("New passwords do not match")
            else:
                st.error("Old password is incorrect")

        if st.button("Go to Login"):
            st.session_state.page = 'login'
            st.experimental_rerun()

        st.markdown("</div>", unsafe_allow_html=True)

    # Function to render the forgot password form
    def forgot_password():
        st.markdown("""
        <style>
        .forgot-password-form {
            max-width: 400px;
```

```
            margin: auto;
            padding: 20px;
            background-color: #f9f9f9;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .forgot-password-form h2 {
            text-align: center;
            color: #004d99;
        }
        .forgot-password-form input {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
        }
        .forgot-password-form button {
            width: 100%;
            padding: 10px;
            background-color: #004d99;
            border: none;
            color: white;
            border-radius: 5px;
            cursor: pointer;
        }
        .forgot-password-form button:hover {
            background-color: #003366;
        }
        </style>
        <div class="forgot-password-form">
            <h2>Forgot Password</h2>
        """, unsafe_allow_html=True)

        phone_number = st.text_input("Enter your phone number", key="forgot_phone_number")
        if st.button("Send OTP", key="forgot_send_otp"):
            send_otp()
        otp = st.text_input("Enter OTP", key="forgot_otp")
        new_password = st.text_input("New Password", type="password", key="forgot_new_password")
        confirm_password = st.text_input("Confirm New Password", type="password", key="forgot_confirm_password")
        if st.button("Reset Password", key="forgot_reset_password"):
            if otp == st.session_state.otp:
                if new_password == confirm_password:
                    st.session_state.password = new_password
                    st.success("Password reset successfully")
                else:
                    st.error("New passwords do not match")
            else:
                st.error("Invalid OTP")

        if st.button("Go to Login"):
            st.session_state.page = 'login'
            st.experimental_rerun()

        st.markdown("</div>", unsafe_allow_html=True)

    # Function to render the selection page
    def selection_page():
        st.markdown("""
        <style>
        .selection-page {
            max-width: 600px;
```

```
            margin: 100px auto;
            padding: 30px;
            background-color: #ffffff;
            border-radius: 20px;
            box-shadow: 0 0 20px rgba(0, 0, 0, 0.2);
            text-align: center;
        }
        .selection-page h2 {
            color: #004d99;
            font-size: 36px;
            font-family: 'Arial', sans-serif;
            position: relative;
            animation: fadeIn 1s ease-in-out;
            animation-delay: 0.5s;
        }
        .selection-page h2::after {
            content: '';
            display: block;
            width: 100%;
            height: 3px;
            background-color: #004d99;
            position: absolute;
            bottom: -10px;
            left: 0;
            transform: scaleX(0);
            transition: transform 0.3s ease-in-out;
        }
        .selection-page h2:hover::after {
            transform: scaleX(1);
        }
        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(-20px); }
            to { opacity: 1; transform: translateY(0); }
        }
        </style>
        <div class="selection-page">
            <h2>Select an option</h2>
        </div>
        """, unsafe_allow_html=True)

        columns = st.columns((20,20))

        with columns[0]:
            sensor_procurement = st.button("Sensor Procurement", use_container_width=True, key="sensor_procurement"
        with columns[1]:
            sensor_replacement = st.button("Sensor Replacement", use_container_width=True, key="sensor_replacement"


        if sensor_procurement:
            st.session_state.track_option = 'sensor_procurement'
            st.session_state.page = 'main'
            st.rerun()

        if sensor_replacement:
            st.session_state.track_option = 'sensor_replacement'
            st.session_state.page = 'main'
            st.rerun()

    # Function to render the main page
    def main():
        # Custom logo and title
        st.write("""
```

```
            <div style="display: flex; flex-direction: column; align-items: center;">
                <img src="https://1000logos.net/wp-content/uploads/2021/08/Reliance-Industries-Limited-RIL-Logo.jpg"
            </div>
            """, unsafe_allow_html=True)

        html_string = '<h1>DTA_DGT-SensorTrack</h1>'
        st.markdown(html_string, unsafe_allow_html=True)
        subheader_text = "Procurement" if st.session_state.track_option == 'sensor_procurement' else "Replacement"
        html_string = f'<h2 class="subheader"><em>Precision Tracking for Seamless Sensor Procurement and Replacement
        st.markdown(html_string, unsafe_allow_html=True)

        if st.button("Logout"):
            st.session_state.logged_in = False
            st.session_state.page = 'login'
            st.rerun()

        with st.expander("How to Use This App"):
            st.markdown("""
                Welcome to DTA_DGT-SensorTrack! Here's a step-by-step guide to help you navigate through the app:

                **1. Upload Your Excel File**
                - Click on the 'Choose an Excel file' button.
                - Select the Excel file from your local storage.

                **2. Filter Your Data**
                - Use the filters on the sidebar to narrow down your data:
                  - **Plant**: Select the plant from the dropdown menu.
                  - **Make**: Choose the make of the sensor.
                  - **Model**: Pick the model of the sensor.
                  - **Gas Type**: Select the type of gas.
                  - **Date Range**: Select the procurement or replacement date range.

                **3. View and Download Filtered Data**
                - The filtered data will be displayed on the main page.
                - Use the checkboxes to show the filtered data, completed procurements/replacements, or remaining pr
                - You can remove columns by selecting them in the multiselect dropdown.
                - Use the 'Download Options' in the sidebar to download the filtered data in Excel or PDF format.

                **4. Manage Your Account**
                - Use the buttons to log out, change password, or reset password if needed.

            """)



        def local_css():
            css = """
            <style>
            body {
                font-family: "Arial", sans-serif;
                background-color: #f5f5f5;
            }
            h1, h2, h3 {
                color: #004d99;
            }
            .subheader {
                margin-right: 100px;  /* Adjust this value to shift more or less */
            }
            </style>
            """
            st.markdown(css, unsafe_allow_html=True)
```

```
        local_css()

        uploaded_file = st.file_uploader("Choose an Excel file", type="xlsx")
        if uploaded_file:
            df = pd.read_excel(uploaded_file)

            # Convert 'Sensor Validity' to datetime
            df['Sensor Validity'] = pd.to_datetime(df['Sensor Validity'], format='%Y-%m-%d', errors='coerce')

            # Ensure 'Sensor Procurement' is numeric and then convert to datetime
            df['Sensor Procurement'] = pd.to_numeric(df['Sensor Procurement'], errors='coerce')
            df['Sensor Procurement'] = pd.to_datetime(df['Sensor Procurement'], origin='1899-12-30', unit='D', error

            # Directly convert 'Sensor Replacement Date' to datetime
            df['Sensor Replacement Date'] = pd.to_datetime(df['Sensor Replacement Date'], format='%m/%d/%Y', errors=

            st.write("## Uploaded Data Preview:")
            st.dataframe(df, height=250, width=1200)

            st.sidebar.header("Filtering Preferences")

            plant = st.sidebar.selectbox("Select Plant", ['All'] + df['PLANT'].unique().tolist())
            make = st.sidebar.selectbox("Select Make", ['All'] + df['MAKE'].unique().tolist())
            model = st.sidebar.selectbox("Select Model", ['All'] + df['MODEL'].unique().tolist())
            gas_type = st.sidebar.selectbox("Select Gas Type", ['All'] + df['GAS TYPE'].unique().tolist())

            date_label = "Sensor Procurement" if st.session_state.track_option == 'sensor_procurement' else "Sensor

            if st.session_state.track_option == 'sensor_replacement':
                replacement_option = st.sidebar.selectbox("Replacement Option", ['See Current Replacements', 'See Fu

                if replacement_option == 'See Current Replacements':
                    date_label = "Sensor Replacement Date"
                    output_text = "Replacements Done"
                else:
                    date_label = "Sensor Validity"
                    output_text = "Upcoming Replacements"


            date_range = st.sidebar.date_input(f"Select {date_label} Range", [])

            filtered_df = df.copy()
            if plant != 'All':
                filtered_df = filtered_df[filtered_df['PLANT'] == plant]
            if make != 'All':
                filtered_df = filtered_df[filtered_df['MAKE'] == make]
            if model != 'All':
                filtered_df = filtered_df[filtered_df['MODEL'] == model]
            if gas_type != 'All':
                filtered_df = filtered_df[filtered_df['GAS TYPE'] == gas_type]

            if date_range:
                if len(date_range) == 2:
                    start_date, end_date = date_range
                    filtered_df = filtered_df[(filtered_df[date_label] >= pd.Timestamp(start_date)) & (filtered_df[d

            if st.session_state.track_option == 'sensor_replacement':
                if replacement_option == 'See Current Replacements':
                    st.sidebar.write(f"### {output_text}: {filtered_df.shape[0]}")
                else:
                    if len(date_range) == 2:
                        st.sidebar.write(f"### {output_text}: {filtered_df.shape[0]}")
```

```
        else:
            st.sidebar.write(f"### Total Procurements Count: {filtered_df.shape[0]}")

        # Making copy of filtered data to avoid errors
        trimmed_filtered_df = filtered_df

        if st.checkbox("Show Filtered Data"):
            # Multiselect dropdown for selecting columns to display or remove
            display_or_exclude = st.radio(
                "Choose whether to display or exclude columns:",
                options=["Display columns", "Exclude columns"],
                key="display_exclude_option"
            )
            if display_or_exclude == "Display columns":
                # Multiselect dropdown for selecting columns to display
                columns_to_display = st.multiselect(
                    "Select columns to display:",
                    options=trimmed_filtered_df.columns.tolist(),
                    key="Filtered_DF"
                )

                if columns_to_display:
                    # Filter the DataFrame to include only the selected columns
                    trimmed_filtered_df = trimmed_filtered_df[columns_to_display]


            else:
                # Multiselect dropdown for selecting columns to remove
                columns_to_remove = st.multiselect(
                    "Select columns to remove:",
                    options=trimmed_filtered_df.columns.tolist()
                )

                # Remove selected columns
                trimmed_filtered_df = trimmed_filtered_df.drop(columns=columns_to_remove)

            # Display the DataFrame
            st.dataframe(trimmed_filtered_df, height=250, width=1200)

        # Calculate the counts based on the selected tracking option
        if st.session_state.track_option == 'sensor_procurement':
            procurement_done_data = filtered_df[filtered_df['procurement done'] == 'Y']
            procurement_left_data = filtered_df[filtered_df['procurement done'] == 'N']
            total_procurement_done = procurement_done_data.shape[0]
            total_procurement_left = procurement_left_data.shape[0]
            st.sidebar.write(f"### Procurements Completed: {total_procurement_done}")
            st.sidebar.write(f"### Procurements Pending: {total_procurement_left}")

            # Making copy of Procurement done data to avoid errors
            trimmed_procurement_done = procurement_done_data

            # Making copy of Procurement left data to avoid errors
            trimmed_procurement_left = procurement_left_data


            if st.checkbox("Show completed procurements"):

                # Multiselect dropdown for selecting columns to display or remove
                display_or_exclude1 = st.radio(
                    "Choose whether to display or exclude columns:",
                    options=["Display columns", "Exclude columns"],
                    key="display_exclude_option1"
```

```
                )
                if display_or_exclude1 == "Display columns":
                    # Multiselect dropdown for selecting columns to display
                    columns_to_display1 = st.multiselect(
                        "Select columns to display:",
                        options=trimmed_procurement_done.columns.tolist(),
                        key="procurements_done1"
                    )

                    if columns_to_display1:
                        # Filter the DataFrame to include only the selected columns
                        trimmed_procurement_done = trimmed_procurement_done[columns_to_display]



                else:
                    # Multiselect dropdown for selecting columns to remove
                    columns_to_remove1 = st.multiselect(
                        "Select columns to remove:",
                        options=trimmed_procurement_done.columns.tolist(),
                        key="procurements_done"
                    )

                    # Remove selected columns
                    trimmed_procurement_done = trimmed_procurement_done.drop(columns=columns_to_remove1)

                # Display the DataFrame
                st.dataframe(trimmed_procurement_done, height=250, width=1200)

            if st.checkbox("Show remaining procurements"):

                # Multiselect dropdown for selecting columns to display or remove
                display_or_exclude2 = st.radio(
                    "Choose whether to display or exclude columns:",
                    options=["Display columns", "Exclude columns"],
                    key="display_exclude_option2"
                )
                if display_or_exclude2 == "Display columns":
                    # Multiselect dropdown for selecting columns to display
                    columns_to_display2 = st.multiselect(
                        "Select columns to display:",
                        options=trimmed_procurement_left.columns.tolist(),
                        key="procurements_left1"
                    )

                if columns_to_display2:
                    # Filter the DataFrame to include only the selected columns
                    trimmed_procurement_done = trimmed_procurement_done[columns_to_display]

                else:
                    # Multiselect dropdown for selecting columns to remove
                    columns_to_remove2 = st.multiselect(
                        "Select columns to remove:",
                        options=trimmed_procurement_left.columns.tolist(),
                        key="procurements_left"
                    )

                    # Remove selected columns
                    trimmed_procurement_left = trimmed_procurement_left.drop(columns=columns_to_remove2)

                # Display the DataFrame
                st.dataframe(trimmed_procurement_left, height=250, width=1200)
```

```python
def convert_df_to_excel(df):
    output = BytesIO()
    writer = pd.ExcelWriter(output, engine='xlsxwriter')

    date_columns = ['Sensor Replacement Date','Sensor Validity', 'Sensor Procurement']
    for col in date_columns:
        if col in df.columns:
            df[col] = df[col].dt.strftime('%Y-%m-%d')

    df.to_excel(writer, index=False, sheet_name='Sheet1')

    workbook  = writer.book
    worksheet = writer.sheets['Sheet1']

    center_format = workbook.add_format({'align': 'center', 'valign': 'vcenter'})

    for column in df:
        column_width = max(df[column].astype(str).map(len).max(), len(column)) + 2
        col_idx = df.columns.get_loc(column)
        worksheet.set_column(col_idx, col_idx, column_width, center_format)

    writer.close()
    processed_data = output.getvalue()
    return processed_data

def convert_df_to_pdf(df):
    date_columns = ['Sensor Validity', 'Sensor Procurement']
    for col in date_columns:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    for col in date_columns:
        if col in df.columns:
            df[col] = df[col].dt.strftime('%Y-%m-%d')

    html = f"""
    <html>
    <head>
        <style>
            table {{
                width: 100%;
                border-collapse: collapse;
            }}
            th, td {{
                border: 1px solid black;
                padding: 5px;
                text-align: left;
                word-wrap: break-word;
            }}
            th {{
                background-color: #f2f2f2;
                text-align: center;
            }}
            td:nth-child({df.columns.get_loc('Sensor Validity') + 1}) {{
                min-width: 100px;
            }}
        </style>
    </head>
    <body>
```

```
            <h1 style="text-align: center;">DTA-DGT_Report</h1>
            {df.to_html(index=False)}
        </body>
        </html>
        """
        path_to_wkhtmltopdf = r'C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe'
        config = pdfkit.configuration(wkhtmltopdf=path_to_wkhtmltopdf)

        pdf = pdfkit.from_string(html, False, configuration=config)
        return pdf

    # Sidebar for download options
    st.sidebar.header("Export Options")

    # Dropdown for selecting data type based on track_option
    if st.session_state.track_option == 'sensor_procurement':
        data_option = st.sidebar.selectbox(
            "Select Data to Download",
            options=[
                "Filtered Data",
                "Procurements Completed",
                "Procurements Pending"
            ]
        )
    elif st.session_state.track_option == 'sensor_replacement':
        data_option = "Filtered Data"

    # Dropdown for selecting file format
    format_option = st.sidebar.selectbox(
        "Select File Format",
        options=["Excel", "PDF"]
    )

    # Prepare the data based on the selected option
    if data_option == "Filtered Data":
        data = trimmed_filtered_df
    if st.session_state.track_option == 'sensor_procurement':
        if data_option == "Procurements Done":
            data = trimmed_procurement_done
        if data_option == "Procurements Left":
            data = trimmed_procurement_left
    elif st.session_state.track_option == 'sensor_replacement':
        if data_option == "Replacements Done":
            data = trimmed_replacement_done
        if data_option == "Replacements Left":
            data = trimmed_replacement_left

    # Convert the data to the selected format
    if format_option == "Excel":
        file_data = convert_df_to_excel(data)
        file_name = 'Filtered DGT Data.xlsx'
        mime_type = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
    else:  # PDF
        file_data = convert_df_to_pdf(data)
        file_name = 'Filtered DGT Data.pdf'
        mime_type = 'application/pdf'

    # Display the download button
    st.sidebar.download_button(
        label=f"Download {format_option} file",
        data=file_data,
        file_name=file_name,
```

```
            mime=mime_type
        )



if __name__ == "__main__":
    if st.session_state.page == 'login':
        login()
    elif st.session_state.page == 'selection':
        selection_page()
    elif st.session_state.page == 'main':
        main()
    elif st.session_state.page == 'change_password':
        change_password()
    elif st.session_state.page == 'forgot_password':
        forgot_password()
```

19. Other Experiences at RIL

Experiences so Far

Central Engineering Services Hands-On Workshop

In this workshop, we got hands-on experience with a variety of industrial instrumentation, including displacement level transmitters, guided wave radar, hydrostatic level transmitters, globe valves, gland packing, multifunction bench (scandura) calibration, proximity sensors, accelerometers, Swagelok tube fittings, Bentley Nevada rotor kits, oil pumps, mass flow meters, Coriolis flow meters, and ultrasonic flow meters.

Prezi

Aromatics Plant Field Visit

The visit was a great success, and the engineers learned a lot about the Aromatics Plant at Reliance Jamnagar. They were impressed by the plant's size and complexity, and they were grateful for the opportunity to learn about the various components and systems involved in the production of aromatics.

Key Takeaways:

1. Control valves are used to regulate the flow of fluids and gases throughout the plant.
2. XVs are used to control the temperature of the various processes.
3. The PIB building houses the plant's control room.
4. Terminal barriers and junction boxes are used to connect the different components of the plant.
5. Transmitters are used to measure and monitor the various parameters of the plant's operations.

Prezi

Central Engineering Services Field Visit

"Day 1: Field Instrumentation Calibration & Troubleshooting Labs :

1. Overview of Field Instrument Calibration Lab Functions
2. Types of Field Instruments & Their Functions
3. Types of Test & Calibration Instruments
4. Application of HART/TREX Communicator
5. Hands-on Configuration Experience with TREX Communicator
6. Hands-on Application of Various Test & Calibration Instruments
7. Types of Fittings Used in Instrumentation Applications
8. Hands-on Calibration Experience with Various Switches, Transmitters, and Gauges

Prezi

Central Engineering Services Field Visit

"Day 2: Valve Maintainence Shop :

* Types of Control Valves & Their Applications
* Valves Overhauling Process
* Overview of Valves' Internal Parts & Their Functions Based on Process Applications
* Important Parameters for Valve Sizing
* Types of Valve Accessories & Their Applications
* Calibration and Configuration of Various Control Valve Positioners (Pneumatic/DVC)
* Hydrotest Rig Functions & Applications
* Overview of Partial Stroke Tests
* Hands-on Experience with Various Control Valve Positioner Configurations & Calibrations

Prezi