LEARNING REPORT

2025

Summarize of learning report from :

Name : Danish Soreng

Roll : 2329181

TRAINING PERIOD:02/05/2025-30/06/2025

Indian Oil Corporation Limited

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY,BHUBANESWAR

Course & Branch : BTech. [Computer Science and communication engineering]

June 18 , 2025

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to **Indian Oil Corporation Limited (IOCL), Barauni Refinery**, for providing me the invaluable opportunity to undertake my industrial training in one of the nation's most prestigious and technically advanced refineries.

I am sincerely thankful to the **Information Systems Department** for their warm reception, constant guidance, and for allowing me to gain first-hand exposure to the digital backbone that supports critical refinery operations. The training has not only enriched my technical understanding but also helped me grow as a responsible and informed engineering student.

I would especially like to extend my heartfelt thanks to **Mr. Arbind Kumar Soreng, GM(IS)**, who acted as my mentor during the training period. Their patience, encouragement, and insightful feedback throughout the internship were instrumental in helping me understand complex systems and successfully complete my assigned project on E-sign application [HSE REPORT PORTAL] & powerbi dashboard [COMPLAINT OVERVIEW]

2025

Indian Oil Corporation Limited

ABOUT INDIAN OIL

A Brief Story About The Company

IndianOil stands as a leading integrated energy company, with operations spanning oil, gas, petrochemicals, and various alternative energy sources. This top Indian Public Sector Undertaking (PSU) holds the 116th position on the Fortune Global 500 list and is recognized for its commitment to innovation, sustainability, and operational excellence.

IndianOil serves over 30 million Indians daily through more than 61,000 retail points, striving for a Net-Zero target by 2046 and aiming for a $1 trillion valuation by 2047. The company commands a significant 45% market share, dominates the LPG sector with its Indane brand, and is expanding its international presence across South Asia and the Middle East. Driven by advanced R&D, robust infrastructure, and a customer-focused growth strategy, IndianOil is dedicated to powering India's energy future with a spirit of passion, care, and trust

VISION

IndianOil's overarching vision is to be "The Energy of India" and a "Globally Admired Company," rooted in its long-standing core values of Care, Innovation, Passion, and Trust (CIPT). To redefine its identity and prepare for the future, IndianOil initiated Project Sattva, a comprehensive company-wide effort that engaged over 20,000 employees to uncover the fundamental essence of what distinguishes IndianOil.

This initiative led to a refreshed set of core values, now including a fifth: Nation-First, embodying the motto "Pehle Indian Phir Oil" (Indian First, then Oil). Each value has been reinterpreted to inspire tangible actions: Care (encompassing Empathy + Vision), Innovation (combining Agility + Foresight), Passion (reflecting Boldness + Dedication), Trust (built on Commitment + Ecosystem Growth), and Nation-First (dedicated to Service to the Nation).

To commemorate these foundational principles, June 30th will now be an annual observance, known as IndianOil Values Day, symbolizing a renewed dedication to these values as the organization moves forward with clear purpose and pride.

MAJOR BUSINESS

IndianOil, a leading Maharatna oil company in India, operates across the entire hydrocarbon value chain, from crude oil refining and transportation to marketing, exploration, petrochemical production, and alternative energy initiatives. The company boasts a global footprint, with subsidiaries in countries such as Sri Lanka, UAE, USA, and the Netherlands, alongside more than 15 joint ventures with respected international partner

01 **Refining: This segment details IndianOil's refining capabilities and the infrastructure utilized for processing crude oil into various usable petroleum products.**

02 **Pipelines: This section outlines IndianOil's extensive pipeline network, which facilitates the transportation of crude oil, refined petroleum products, and gas throughout the country.**

03 **Research & Development: This highlights the company's innovations, technological advancements, and sustainable energy solutions, driven by its R&D initiatives.**

04 **Marketing: This explains IndianOil's strategies and the infrastructure in place for the distribution and sale of fuels, lubricants, and other related products.**

MAJOR BUSINESS

05 **Petrochemicals: This emphasizes IndianOil's involvement in the production of petrochemical products, which are vital components in plastics, various chemicals, and manufacturing processes.**

06 **Natural Gas: This covers the company's expanding portfolio in natural gas, including its sourcing, distribution methods, and future development plans.**

07 **City Gas Distribution (CGD): This specifically focuses on the supply of natural gas to residential households, vehicles, and industrial consumers within urban areas.**

08 **E&P (Exploration & Production): This describes IndianOil's endeavors in exploring for and extracting crude oil and natural gas, both within India and internationally.**

09 **Explosives: This section details operations related to the manufacturing and supply of industrial explosives, primarily used in mining and infrastructure development projects.**

10 **Cryogenics: This involves the technologies and infrastructure dedicated to the production and handling of liquefied gases at extremely low temperatures, with a particular focus on LNG.**

PROJECT & FUTURE FOCUS

IndianOil is currently undertaking over 140 projects, valued at approximately ₹2.5 lakh crore, aimed at strengthening its refining, petrochemical, and marketing infrastructure. The company plans to boost its petrochemical output, expand its pipeline networks, increase access to LPG, and enhance upstream oil exploration efforts.

Furthermore, IndianOil is significantly investing in green energy initiatives, including ethanol blending, biofuels, compressed biogas, green hydrogen, and renewable energy, with a target of 31 GW by 2030. Notable projects include the establishment of a green hydrogen plant at Panipat, the expansion of the Ennore LNG Terminal, and the development of EV infrastructure.

Through continuous innovation and a strong commitment to sustainability, IndianOil is working to ensure energy security, reduce emissions, and support India's transition towards a more environmentally friendly energy future.

REFINERIES DIVISION

Coperate Office
3079/3, Sadiq Nagar, J.B. Tito Marg, New Delhi - 110 049

Registered Office
IndianOil Bhavan, G-9, Ali Yavar Jung Marg, Bandra (East), Mumbai - 400 051

Refinery Division Head Quarter SCOPE Complex, Core-2 7, Institutional Area, Lodhi Road New Delhi - 110003

AOD-Refinery DIGBOI
Refinery PO- DIGBOI - 786171 (Assam)

REFINERIES DIVISION

Barauni Refinery P.O. Barauni Refinery, Dist. Begusarai -86114 (Bihar)

Gujarat Refinery P.O. Jawahar Nagar Dist. Vadodara -391320 (Gujarat)

Guwahati Refinery P.O. Noonmati Guwahati-781020 (Assam)

Haldia Refinery P.O. Haldia Refinery Dist. Midnapur-721606, (West Bengal)

REFINERIES DIVISION

Mathura Refinery P.O.
Mathura Refinery Mathura
-281005, (Uttar Pradesh)

Panipat Refinery P.O.
Panipat Refinery, Panipat-
132140 (Haryana)

Bongaigaon Refinery P.O.
Dhaliagaon Dist. Chirang,
Assam - 783385

Paradip Refinery P.O.
Paradip At. Po. Jhimani -
Via Kujang, Dist.:
Jagatsinghpur, Odisha-
754141

GROUP DIVISIONS

Chennai Petroleum Corporation Ltd. 536, Anna Salai, Teynampet, Chennai - 600018

IndianOil (Mauritius) Ltd Mer Rouge, Port Louis Mauritius

Lanka IOC PLC Lanka IOC Head Office Level 20, West Tower, World Trade Center, Echelon Square, Colombo - 01, Sri Lanka.

IOC Middle East FZE LOBBY 12114, Jebel Ali Free Zone, P.O.Box: 261338, Dubai UAE

ABOUT BARAUNI REFINARY

Introduction and LocationThe Barauni Refinery, inaugurated in 1965 and dedicated to the nation in 1964, is situated on the northern banks of the sacred Ganges River in Begusarai, Bihar. Its strategic location, approximately 125 km from Patna, places it at the intersection of the Eastern and North Eastern Railways. Constructed through a collaborative effort with the Soviet Union and Romania, the refinery commenced operations with an initial investment of roughly ₹49.4 crore.

Capacity Expansion and UpgradationInitially designed to process 1.0 MMTPA of sweet crude from Assam, the refinery's capacity was later expanded to 3.0 MMTPA with the commissioning of AVU-II and AU-III units.
Subsequent revamps in 1985, 1998–99, and 2000 further increased its capacity to 6.0 MMTPA. The Barauni Expansion Project (BXP) in 2002 allowed the refinery to process high-sulfur crude by incorporating major new units such as RFCCU, DHDT, SRU, ARU, and HGU.

DIFFERENT UNITS IN BARAUNI REFINERY

The Barauni Refinery is composed of various integrated processing and utility units designed for efficient crude oil refining and the production of clean fuels:

- **Crude & Vacuum Distillation Units (CDU/VDU): These units are responsible for separating crude oil into various useful fractions.**
- **Hydrocracker Unit (HCU): This unit converts heavy oils into higher-value products such as diesel.**
- **Delayed Coking Unit (DCU): This unit processes residue to generate petroleum coke and lighter fuel components.**
- **Reforming Unit (CRU): This unit works to improve the octane rating of gasoline.**
- **Diesel Hydrotreater (DHDT): This unit desulfurizes diesel, ensuring compliance with BS-VI standards.**
- **Sulfur Recovery Unit (SRU): This unit recovers elemental sulfur, thereby reducing emissions.**
- **Utilities & Offsite Units: These units provide essential support for operations, including power generation, steam production, water treatment, and product storage.**

Indian Oil Corporation Limited

REPORT FROM :

DANISH SORENG.

PROJECT REPORT

E-SIGN APPLICATION

For HSE Report

POWER-BI PROJECT

Complaint Monitoring Data

Indian Oil Corporation Limited

PROJECT OVERVIEW

E-SIGN APPLICATION

Project Statement :

This project delivers a specialized desktop application, the HSE Report Portal, designed to streamline the process of managing and authenticating Health, Safety, and Environment (HSE) reports. By integrating PDF viewing capabilities with digital signature application, it provides a secure and efficient means for officers to review, sign, and archive critical documents, thereby enhancing workflow integrity and compliance.

Goals

The primary goals of this application are:

- To provide an intuitive graphical user interface for easy interaction.
- To enable seamless loading and viewing of PDF documents.
- To facilitate the retrieval and application of e-signatures from a centralized database.
- To allow precise placement of signatures on PDF pages.
- To ensure the secure saving of signed PDF documents.
- To improve the digital workflow for HSE report approval and documentation.

Tools and Applications Used

The development of the HSE Report Portal application leveraged the following key technologies and tools:

- Programming Language: Python
- GUI Framework: Tkinter (Standard Python interface for Tk GUI toolkit)
- PDF Manipulation: PyMuPDF (fitz) for PDF rendering and page interaction.
- PDF Merging: PyPDF2 and ReportLab for digitally embedding signatures into PDF documents.
- Database Management: MySQL for storing officer details and their digital signatures.
- Image Processing: Pillow (PIL) for handling and resizing signature images.
- Version Control: (Assumed, typically Git for collaborative development)
- Operating System: (Assumed, typically Windows, macOS, or Linux for desktop applications)

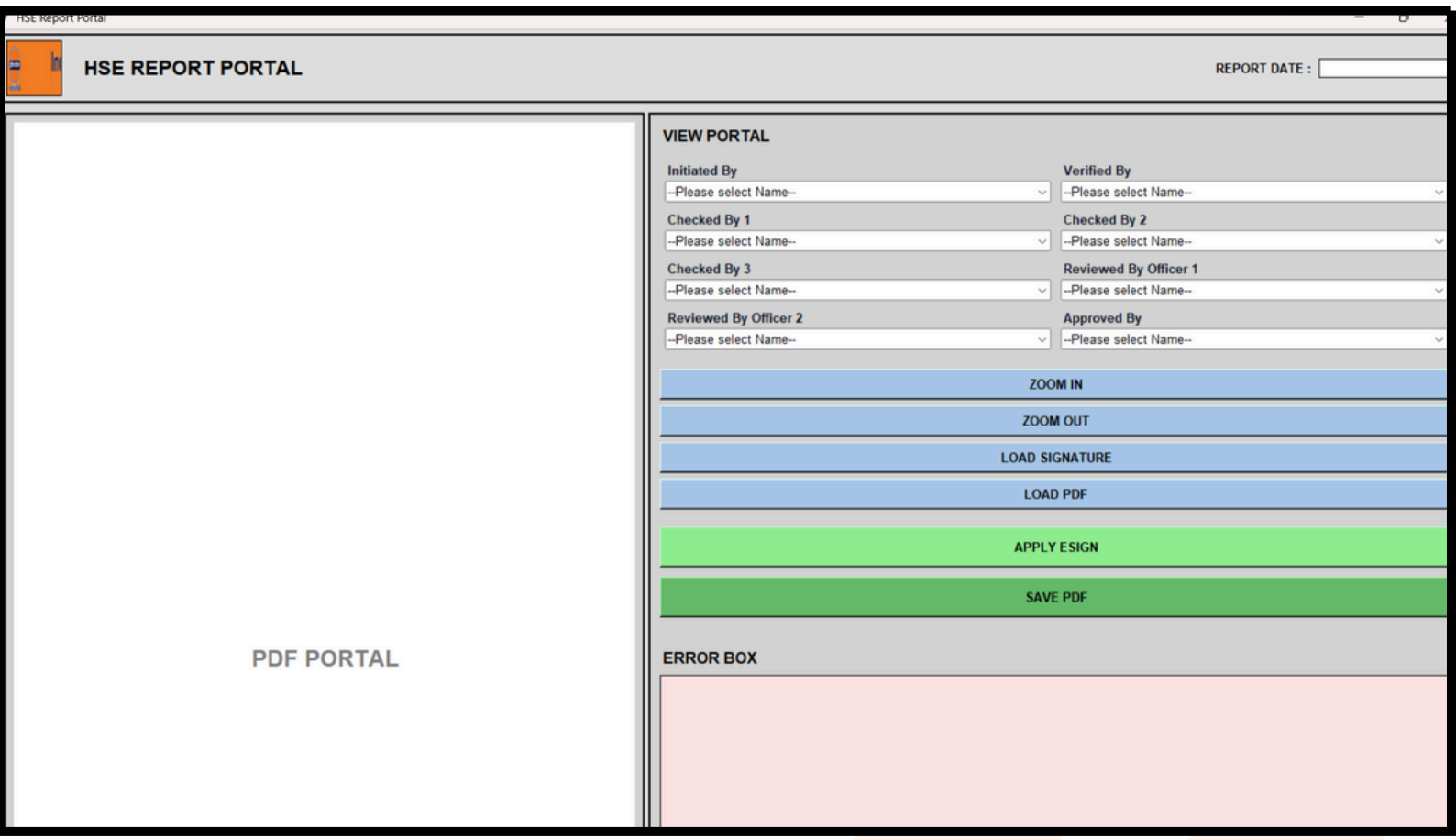
HSE PDF PORTAL

This report provides an in-depth and easy-to-understand explanation of the copiolet_test.py program, breaking down its functionalities, technologies used, and core components.

HSE Report Portal Application Explanation

The copiolet_test.py program develops a HSE (Health, Safety, and Environment) Report Portal Application using the Tkinter library for its graphical user interface (GUI). This application is designed to allow users to load PDF documents, view them, select officers from a database, load their e-signatures, place these signatures onto the PDF, and then save the PDF with the applied e-signatures. It integrates with a MySQL database to manage officer names and their corresponding digital signatures.

HSE PDF PORTAL



LIBRARY USED

- io: This module provides tools for working with I/O streams, used here for handling image data in memory (e.g., converting bytes to an image).
- re: The regular expression module, used for pattern matching in strings, specifically for extracting dates from filenames.
- pathlib.Path: A modern way to handle file system paths, making path manipulation more intuitive and robust.
- datetime: Used for working with dates and times, specifically for parsing and formatting report dates.
- tkinter: The standard Python interface to the Tk GUI toolkit. It provides widgets like Tk (main window), Canvas (for drawing PDF), Text (for displaying text/errors), Button, Label, Frame (for layout), Scrollbar, messagebox (for pop-up messages), END (text widget constant), and filedialog (for file open/save dialogs).
- tkinter.ttk.Combobox: A themed Tkinter widget for dropdown selection.
- PIL (Pillow): The Python Imaging Library fork. Image is used for opening, manipulating, and resizing images (e.g., signatures). ImageTk is used to convert PIL images into Tkinter-compatible photo images for display.
- mysql.connector: This library is essential for connecting to and interacting with a MySQL database, specifically to fetch officer names and their e-signatures.
- fitz (PyMuPDF): A powerful library for working with PDF documents. It's used for opening PDFs, rendering pages to images, and getting page dimensions.
- OUTPUT_PATH, ASSETS_PATH: These Path objects define where the script is located and where UI assets (like the IndianOil logo) are expected to be found. ASSETS_PATH specifically points to a folder containing image assets for the GUI.

```
import io
import re
from pathlib import Path
from datetime import datetime
from tkinter import (
    Tk, Canvas, Text, Button, Label, Frame, Scrollbar, messagebox, END, filedialog
)
from tkinter.ttk import Combobox
from PIL import Image, ImageTk
import mysql.connector
import fitz  # PyMuPDF

# Define the path for assets. Please update this path
# to the correct directory where your 'image_1.png' (IndianOil logo) is located.
# Example: ASSETS_PATH = Path("C:/Your/Path/To/Assets")
OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(r"C:\DataAnalysis\Projects\pdfreader\TkinterConve
```

CLASS INITIALIZATION

This is the constructor method for the main application class. It sets up the main window and initializes all the instance variables that will store the application's state and data.

- self.root: The main Tkinter window.
-
- Window properties: Title, size (geometry), background color (bg), and resizable property are set.
-
- grid_rowconfigure/grid_columnconfigure: These lines configure how rows and columns in the main window's grid layout will behave when the window is resized. weight=1 means they will expand to fill available space.
-
- Variables Section: A comprehensive set of instance variables is initialized to None, 0, 1.0, or empty data structures. These variables manage the state of the application, including the loaded PDF, current zoom, page number, signature data, drag/pan state, caches, and UI element references.

```
class HSEReportPortalApp:
    def __init__(self, root):
        self.root = root
        self.root.title("HSE Report Portal")
        self.root.geometry("1160x780")
        self.root.configure(bg="#E0E0E0")
        self.root.resizable(True, True)

        self.root.grid_rowconfigure(0, weight=0)
        self.root.grid_rowconfigure(1, weight=1)
        self.root.grid_columnconfigure(0, weight=1)

        ###
        # Variables
        ###
        self.pdf_path = None # Stores the file path of the currently loaded PDF.
        self.pdf_img_tk = None # Tkinter PhotoImage of the current PDF page for di
        self.pdf_doc = None # The fitz (PyMuPDF) document object of the loaded PDF
```

HELPER FUNCTIONS FOR UI STYLING

These functions assist in creating visually consistent buttons and handling color transformations for hover effects.

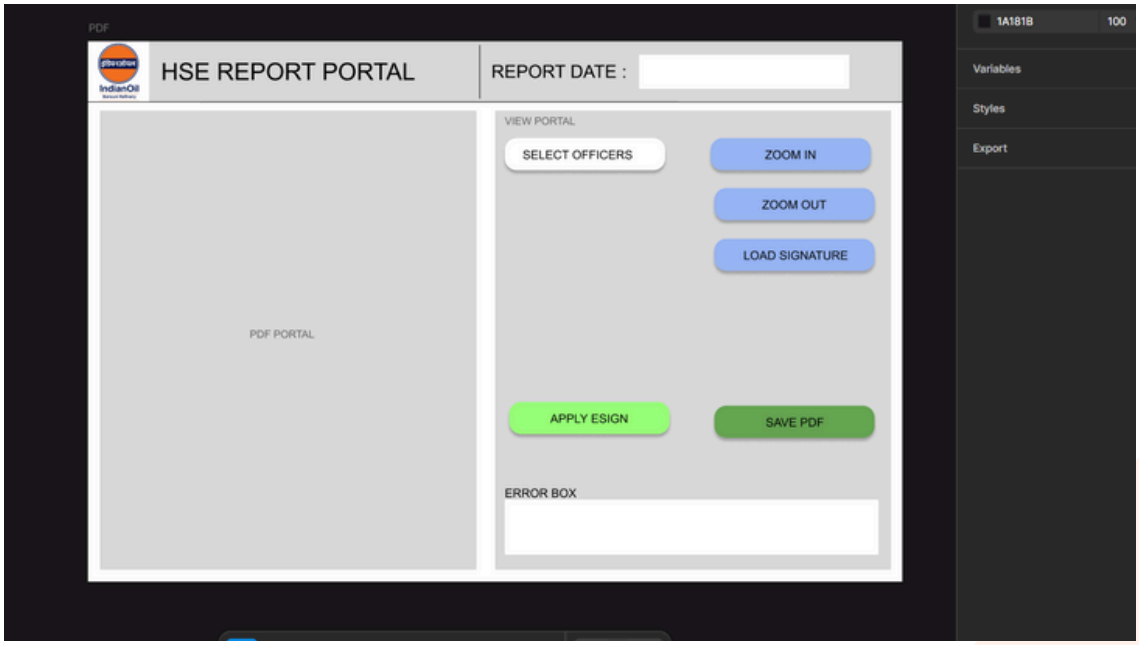
- self.root: The main Tkinter window.
-
- Window properties: Title, size (geometry), background color (bg), and resizable property are set.
-
- grid_rowconfigure/grid_columnconfigure: These lines configure how rows and columns in the main window's grid layout will behave when the window is resized. weight=1 means they will expand to fill available space.
-
- Variables Section: A comprehensive set of instance variables is initialized to None, 0, 1.0, or empty data structures. These variables manage the state of the application, including the loaded PDF, current zoom, page number, signature data, drag/pan state, caches, and UI element references.

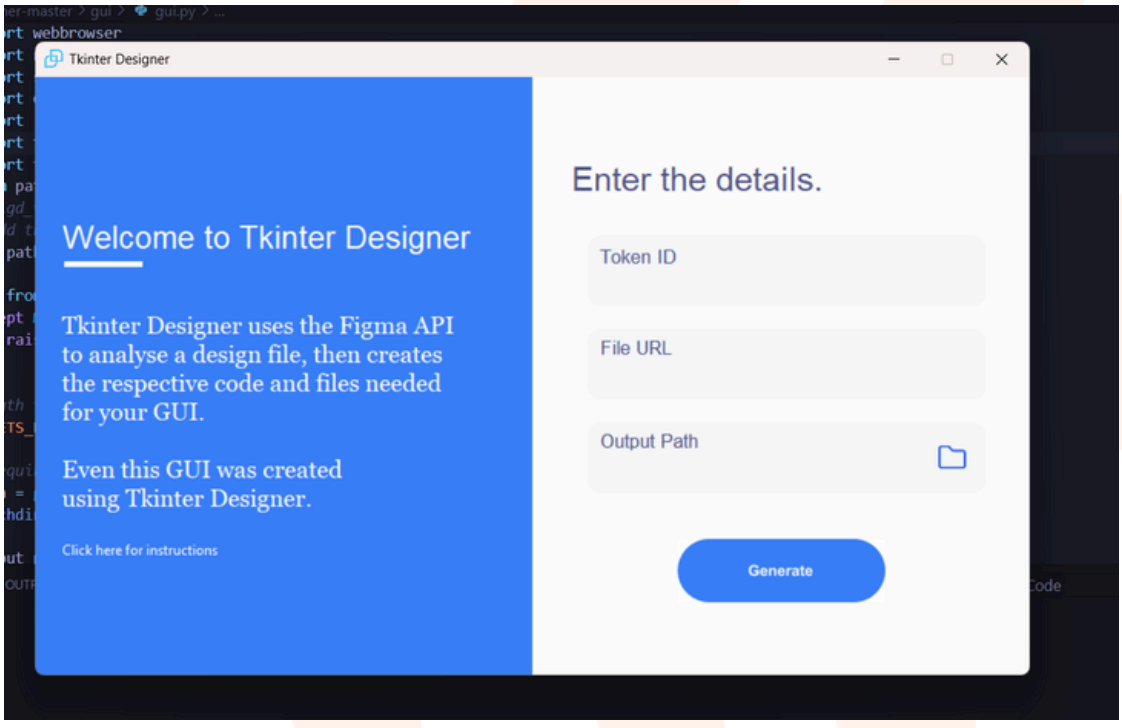
```
    def _darken_color(self, hex_color, percent):
        """Darkens a hex color by a given percentage."""
        hex_color = hex_color.lstrip('#')
        rgb = tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))

        darkened_rgb = tuple(max(0, int(c * (1 - percent / 100))) for c in rgb)
        return '#%02x%02x%02x' % darkened_rgb

    def _lighten_color(self, hex_color, percent):
        """Lightens a hex color by a given percentage."""
        hex_color = hex_color.lstrip('#')
        rgb = tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))

        lightened_rgb = tuple(min(255, int(c * (1 + percent / 100))) for c in rgb)
        return '#%02x%02x%02x' % lightened_rgb

    def _on_button_enter(self, button, original_bg_color):
        """Changes button color on mouse enter."""
        button.config(bg=self._lighten_color(original_bg_color, 15))

    def _on_button_leave(self, button, original_bg_color):
```

UI BUILDING

This extensive method is responsible for constructing the entire graphical user interface of the application, arranging various widgets like labels, frames, buttons, dropdowns, and the PDF display canvas.

- The UI is build from Extension Known as Figma to tkinter Conversion
- This Initially Helps to build UI using drag and drop box methods .
- Allowing us to Import Figma Designs To convert Them Into tkinter GUI.





UI ELEMETS

- Frames: The UI is structured using Frame widgets (top_bar_frame, main_content_frame, frame_pdf, controls_frame, officer_buttons_frame, error_frame) to organize components into logical sections.

- Layout Managers: grid() is primarily used for arranging frames and widgets in a row-and-column structure, while pack() is used for simple stacking or filling within a frame.

- IndianOil Logo: Attempts to load an animated GIF logo (animated_logo.gif). If not found or if there's an error, it falls back to a static PNG (image_1.png). If both fail, it displays text. The _animate_logo method (detailed below) handles cycling through GIF frames.

- Labels: Label widgets display static text, such as "HSE REPORT PORTAL," "REPORT DATE :," "VIEW PORTAL," and "ERROR BOX."

- report_date_entry: A Text widget, initially disabled, used to display the report date, which is extracted from the loaded PDF's filename.

UI ELEMETS

- canvas_pdf: This is a Canvas widget where the PDF pages are rendered as images. It's configured to be scrollable and handles mouse events for drag-to-pan functionality. An initial "PDF PORTAL" text is displayed.

- Officer Dropdowns: A loop creates Label and Combobox widgets for various officer roles (e.g., "Initiated By," "Approved By"). These dropdowns are populated with names from the database.

- Buttons: Several buttons (ZOOM IN, ZOOM OUT, LOAD SIGNATURE, LOAD PDF, APPLY ESIGN, SAVE PDF) are created using the create_rounded_button helper, each linked to a specific application function.

- Error Box: A Text widget (self.error_text) within self.error_frame is used to display error messages or status updates to the user. It's disabled by default (read-only) and has a scrollbar.

CORE PROGRAMM

To Initiate Our program we must Connect Database [Mysql] to fetch e-sign and execute sql querries

fetch_names(): Connects to a MySQL database (configured with host="localhost", user="root", password="password", database="project"). It queries the iocl table to SELECT name and populates the officer dropdowns (Combobox widgets) with these names, along with a default "Please select Name" option.

```
def fetch_names(self):
    try:
        conn = mysql.connector.connect(host="localhost", user="root", password=
        cursor = conn.cursor()
        cursor.execute("SELECT name FROM iocl ORDER BY empid ASC")
        result = cursor.fetchall()
        all_names = [row[0] for row in result]
        conn.close()

        name_index = 0
        for role_title in self.officer_roles_config:
            dropdown = self.officer_dropdowns[role_title]
            options_for_dropdown = ["--Please select Name--"]
            if name_index < len(all_names):
```

These are additional Functions for

animated logo (top left) ,

Recenter pdf portal text (ensures when there is no pdf loaded in canvas the text still remains display inside the box)

Log Error (This is a utility function to update the self.error_text widget.)

```
def _animate_logo(self):
    """Cycles through GIF frames for the logo animation."""
    if self.logo_frames:
        self.logo_frame_index = (self.logo_frame_index + 1) % len(self.logo_fra
        self.logo_label.config(image=self.logo_frames[self.logo_frame_index])
        self.logo_animation_id = self.root.after(self.animation_speed_ms, self

def _recenter_pdf_portal_text(self, event):
    # Recenter the "PDF PORTAL" text when the canvas is resized, if no PDF is
    if not self.pdf_doc and self.canvas_pdf.find_withtag("pdf_portal_text"):
        self.canvas_pdf.coords("pdf_portal_text", event.width / 2, event.heigh

def log_error(self, msg):
```

CORE PROGRAMM

set report date filename(path): Extracts a date from the filename of the loaded PDF using a regular expression (re.search). If a date in the format hse_report_DDMMYYYY is found, it parses and formats it as DD:MM:YYYY and displays it in the report_date_entry text field. Otherwise, it shows "Unknown."

```
def set_report_date_from_filename(self, path):
    filename = Path(path).stem
    m = re.search(r"hse_report[_]?(\d{2})(\d{2})(\d{4})", filename, re.I)
    if m:
        day, month, year = m.groups()
        try:
            dt = datetime.strptime(f"{day}{month}{year}", "%d%m%Y")
            formatted = dt.strftime("%d:%m:%Y")
        except Exception:
            formatted = f"{day}:{month}:{year}"
        self.report_date_entry.configure(state="normal")
        self.report_date_entry.delete(1.0, 'end')
        self.report_date_entry.insert('end', formatted)
        self.report_date_entry.configure(state="disabled")
```

Load pdf Function:

- Opens a file dialog to allow the user to select a PDF file.
- Uses fitz.open(path) to load the PDF document.
- Calculates an initial self.current_zoom level to ensure the entire PDF page fits within the canvas_pdf when loaded.
- Calls render_pdf_page() to display the first page.
- Calls set_report_date_from_filename() to attempt to set the report date.

```
def load_pdf(self):
    path = filedialog.askopenfilename(filetypes=[("PDF Files", "*.pdf")])
    if not path:
        return
    try:
        self.pdf_path = path
        self.pdf_doc = fitz.open(path)
        self.current_page_num = 0
        self.root.update_idletasks() # Ensure canvas has correct dimensions
        page = self.pdf_doc.load_page(self.current_page_num)

        pdf_width_pts = page.rect.width
```

CORE PROGRAMM

Rendering PDF Pages(imp):

- This is a crucial function that handles displaying the current PDF page on the canvas_pdf.
- It uses fitz.Matrix and page.get_pixmap() to render the PDF page as an image (pix) at the current zoom level.
- The pix data is then converted into a PIL Image and finally into a ImageTk.PhotoImage for display on the Tkinter Canvas.
- It clears any previous content on the canvas and draws the new PDF image.
- Crucially, it also iterates through self.placed_signatures for the current page and draws preview images of already "applied" signatures, complete with a dashed gray border.
- It displays a small preview of the currently active signature (if any) in the top-right corner of the canvas.

```
def render_pdf_page(self):
    if not self.pdf_doc:
        # If no PDF is loaded, ensure "PDF PORTAL" text is shown
        if not self.canvas_pdf.find_withtag("pdf_portal_text"):
            self.canvas_pdf.create_text(
                self.canvas_pdf.winfo_width() / 2, self.canvas_pdf.winfo_height
                text="PDF PORTAL", font=("Inter", 18, "bold"), fill="#808080",
            )
        return

    self.canvas_pdf.delete("pdf_portal_text")
    self.canvas_pdf.delete("sig_preview_rect")
    self.canvas_pdf.delete("sig_preview_img")
    self.canvas_pdf.delete("sig_display_top_right")
```

- **Zoom in-out Function** : These methods adjust the self.current_zoom level (by multiplying or dividing by 1.25) and then call render_pdf_page() to re-render the PDF at the new zoom. They also attempt to keep the view centered.

```
def zoom_in(self):
    if self.pdf_doc and self.current_zoom < 4.0:
        self.current_zoom *= 1.25
        self.render_pdf_page()
        # Adjust view after zooming to keep content centered
        self.canvas_pdf.xview_moveto(self.canvas_pdf.xview()[0] / 1.25)
        self.canvas_pdf.yview_moveto(self.canvas_pdf.yview()[0] / 1.25)

def zoom_out(self):
```

CORE PROGRAMM

Register E-Sign According to mouse click: These three methods(on_drag_start()/ on_drag_motion()/on_drag_end()) implement the "drag-to-pan" functionality on the PDF canvas:

- on_drag_start: Records the initial mouse click position and changes the cursor to a "hand."
- on_drag_motion: If a drag threshold is exceeded, it continuously scrolls the canvas_pdf based on the mouse movement, creating the panning effect.
- on_drag_end: Resets the cursor. If the mouse movement was below the drag_threshold, it's considered a "click," and _place_signature_on_click() is called.

```
def on_drag_start(self, event):
    """Records the starting coordinates for dragging and changes cursor."""
    self.drag_start_x = event.x
    self.drag_start_y = event.y
    self.is_dragging = False # Reset drag flag
    self.canvas_pdf.config(cursor="hand2") # Change cursor to hand


def on_drag_motion(self, event):
    """Drags the canvas view based on mouse motion."""
    if self.drag_start_x is not None and self.drag_start_y is not None:
        # Check if motion exceeds threshold to confirm a drag
        if not self.is_dragging and (abs(event.x - self.drag_start_x) > self.drag_threshold or \
                                     abs(event.y - self.drag_start_y) > self.drag_threshold):
            self.is_dragging = True

        if self.is_dragging:
            dx = event.x - self.drag_start_x
            dy = event.y - self.drag_start_y

            # Scroll the canvas content
            self.canvas_pdf.xview_scroll(-dx, "units")
            self.canvas_pdf.yview_scroll(-dy, "units")
```

Place signature on click (event):

- When the user clicks on the PDF canvas (and it's not a drag), this method is called.
- It calculates the PDF coordinates (in points) where the signature should be placed, taking into account the current zoom level and the difference in origin between Tkinter (top-left) and PDF (bottom-left).
- It stores this position in self.signature_position_pdf.
- It then draws a temporary red-bordered preview of the active signature on the canvas at the clicked location.

```
def _place_signature_on_click(self, event):
    """Helper to place signature if the mouse event was a click (not a drag)."""
    if not self.pdf_doc or not self.current_active_signature_data['pil_img']:
        self.log_error("Load a PDF and select/load a signature to place it.")
        return

    canvas_click_x = self.canvas_pdf.canvasx(event.x)
    canvas_click_y = self.canvas_pdf.canvasy(event.y)

    page = self.pdf_doc.load_page(self.current_page_num)
    page_height_pdf_pts = page.rect.height

    pdf_x_top_left_click = canvas_click_x / self.current_zoom
    pdf_y_top_left_click = canvas_click_y / self.current_zoom

    sig_pil = self.current_active_signature_data['pil_img']
    sig_target_width_pt = 120
    sig_pil_aspect_ratio = sig_pil.height / sig_pil.width
    sig_target_height_pt = sig_target_width_pt * sig_pil_aspect_ratio
```

CORE PROGRAMM

Username Select from dropdown(event): This callback is triggered when a user selects a name from any of the officer dropdowns. It updates self.active_name_for_signature_load with the selected name and provides a hint in the error box.

```
def _on_officer_dropdown_selected(self, event):
    """
    Callback for any of the new officer role dropdowns being selected.
    Sets the active_name_for_signature_load based on the most recent selection.
    """
    selected_dropdown_widget = event.widget
    selected_name = selected_dropdown_widget.get()

    # Only update active_name_for_signature_load if a valid name is selected, not the placeholder
    if selected_name != "--Please select Name--":
        self.active_name_for_signature_load = selected_name
        self.log_error(f"'{selected_name}' selected. Click 'LOAD SIGNATURE' to load this person's e-sign.")
    else:
        self.active_name_for_signature_load = None
        self.log_error("Please select a valid name to load a signature.")


    # Optionally clear current active signature preview in top-right if a new name is selected
    self.current_active_signature_data = {'pil_img': None, 'tk_img': None, 'name': None, 'image_bytes': None}
    self.canvas_pdf.delete("sig_display_top_right")
```

Load Signature:

- Retrieves the selected name from self.active_name_for_signature_load.
- Checks if the signature is already in self.loaded_signatures_cache to avoid redundant database queries.
- If not cached, it connects to the MySQL database and queries the iocl table to SELECT ESIGN (the signature image bytes) for the selected name.
- It converts the retrieved bytes into a PIL Image and then a Tkinter PhotoImage for a small top-right preview.
- Stores the PIL image, Tkinter image, and original bytes in self.loaded_signatures_cache and sets self.current_active_signature_data.
- Calls render_pdf_page() to update the UI with the active signature preview.

```
def load_signature(self):
    if not name:
        self.log_error("Please select an officer from one of the dropdowns first to load their signature.")
        return

    if name in self.loaded_signatures_cache:
        self.log_error(f"Signature for '{name}' is already loaded. Activated from cache. Click on PDF to place.")
        cached_data = self.loaded_signatures_cache[name]
        self.current_active_signature_data = {
            'pil_img': cached_data['pil_img'],
            'tk_img': cached_data['tk_img'],
            'name': name,
            'image_bytes': cached_data['image_bytes']
        }
        self.render_pdf_page()
        return

    try:
        conn = mysql.connector.connect(host="localhost", user="root", password="password", database="project")
        cursor = conn.cursor()
        cursor.execute("SELECT ESIGN FROM iocl WHERE name=%s", (name,))
        result = cursor.fetchone()
        cursor.close()
```

CORE PROGRAMM

Apply The Signature:

- This method is called when the "APPLY ESIGN" button is clicked.
- It takes the pil_img, calculated position (from _place_signature_on_click), and dimensions of the current_active_signature_data.
- It stores this information in the self.placed_signatures dictionary, categorized by page number. This list of placed signatures is what will actually be merged into the PDF when saved.
- It then clears the temporary signature preview and calls render_pdf_page() to redraw the page with the "permanently" placed signature preview.

```
def apply_signature(self):
    if not self.pdf_doc:
        self.log_error("No PDF loaded.")
        return
    if not self.signature_position_pdf:
        self.log_error("Click on the PDF to set signature position before applying.")
        return
    if not self.current_active_signature_data['pil_img']:
        self.log_error("No active signature selected/loaded.")
        return

    active_pil_img = self.current_active_signature_data['pil_img']
    active_name = self.current_active_signature_data['name']

    sig_target_width_pt = 120
    sig_pil_aspect_ratio = active_pil_img.height / active_pil_img.width
    sig_target_height_pt = sig_target_width_pt * sig_pil_aspect_ratio

    # Store the placed signature information
    if self.current_page_num not in self.placed_signatures:
        self.placed_signatures[self.current_page_num] = []

    self.placed_signatures[self.current_page_num].append({
        'pil_img': active_pil_img,
        'position': self.signature_position_pdf, # Already bottom-left origin for actual merge
        'target_width_pt': sig_target_width_pt,
        'target_height_pt': sig_target_height_pt,
        'name': active_name
    })

    self.signature_position_pdf = None # Clear placement position after 'applying'
    self.log_error(f"Signature '{active_name}' placed on page {self.current_page_num + 1}. Click on PDF to place another or 'SAVE PDF'.")
    self.render_pdf_page() # Re-render to show the permanently placed signature preview
```

CORE PROGRAMM

Saving The Pdf:

- This is the critical function for merging signatures into the PDF.
- It prompts the user to choose a save location and filename using filedialog.asksaveasfilename.
- It uses PyPDF2.PdfFileReader to read the original PDF and PyPDF2.PdfFileWriter to create a new PDF.
- Finally, writer.write(output_file) saves the new PDF with all the embedded signatures.
- After saving, it reloads the newly saved PDF into fitz and displays it in the application, and clears self.placed_signatures.
- Includes error handling and suggestions for installing PyPDF2 and ReportLab if issues occur.

```
def save_pdf(self):
    if not self.pdf_doc:
        self.log_error("No PDF to save.")
        return
    if not self.placed_signatures:
        self.log_error("No signatures have been placed on the PDF yet. Nothing to save.")
        return

    save_path = filedialog.asksaveasfilename(
        defaultextension=".pdf",
        filetypes=[("PDF Files", "*.pdf")],
        title="Save PDF as"
    )
    if not save_path:
        return

    try:
        # Using PdfFileReader and PdfFileWriter for compatibility with older PyPDF2 (<= 2.x.x)
        from PyPDF2 import PdfFileReader, PdfFileWriter
        from reportlab.pdfgen import canvas as rp_canvas
        from reportlab.lib.utils import ImageReader

        original_pdf_bytes = self.pdf_doc.tobytes()
        reader = PdfFileReader(io.BytesIO(original_pdf_bytes))
```

CORE PROGRAMM

Main Execution Block

This standard Python construct ensures that the main() function is called only when the script is executed directly, not when it's imported as a module.

```
def main():
    root = Tk()
    app = HSEReportPortalApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

- main(): Creates the main Tkinter window (Tk()), instantiates the HSEReportPortalApp class, and starts the Tkinter event loop (root.mainloop()), which keeps the GUI running and responsive to user interactions.
- if __name__ == "__main__":: This is a common Python idiom that ensures the main() function is called only when the script is run as the primary program.

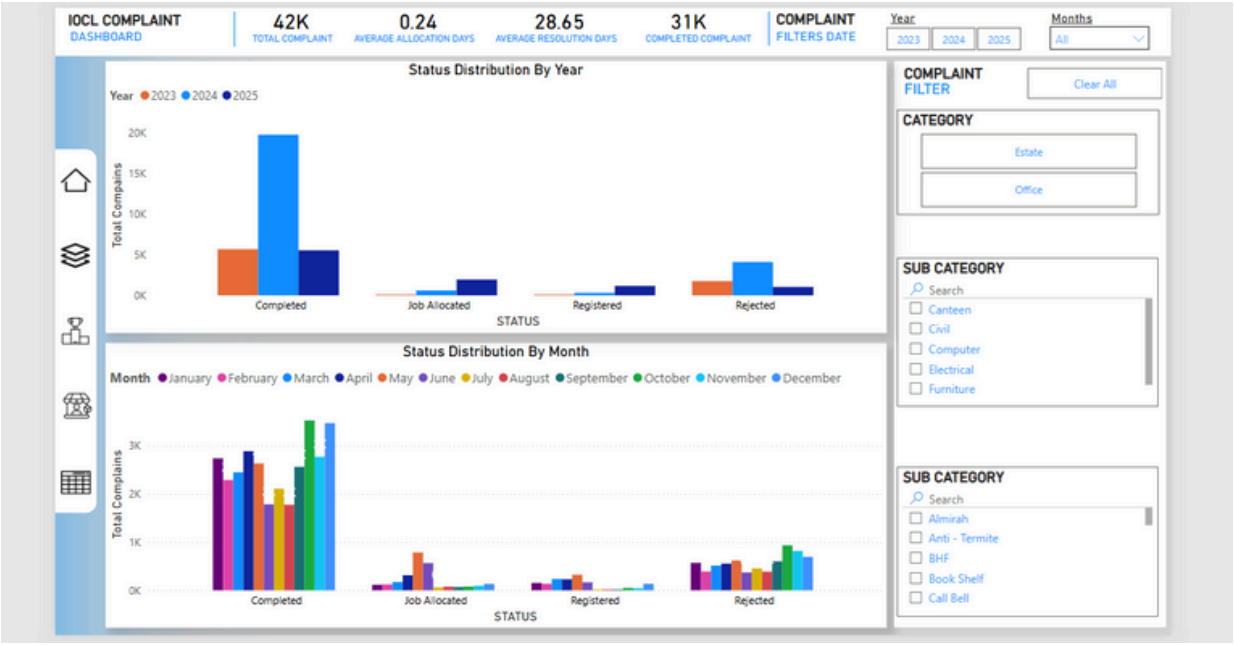
POWERBI PROJECT

A Brief About Powerbi

Microsoft Power BI is a leading business intelligence (BI) tool that empowers organizations to transform raw data from various sources into meaningful, interactive, and visually immersive insights. It's a suite of software services, apps, and connectors designed to help users, from business analysts to executives, understand their data, track key performance indicators (KPIs), and make data-driven decisions.

Complaint Dashboard

Preview :



PROJECT OVERVIEW

This project aims to develop an interactive and insightful Power BI dashboard for the IOCL Estate Office complaint management system. The primary goal is to provide a clear, real-time, and actionable overview of complaint trends, resolution efficiency, and performance metrics. This will enable better decision-making, resource allocation, and ultimately, improved user satisfaction.

Key Goals:

- Enhance Visibility: Provide a comprehensive view of all complaints, their statuses, categories, and sub-categories.
- Track Performance: Monitor key performance indicators (KPIs) such as total complaints, average allocation days, average resolution days, and completed complaints.
- Identify Bottlenecks: Pinpoint areas of inefficiency, common complaint types, and underperforming vendors or employees.
- Improve Accountability: Create leaderboards to recognize top performers and highlight areas needing improvement among employees and vendors.
- Support Data-Driven Decisions: Empower the Estate Office with data to optimize processes, allocate resources effectively, and prioritize complaint resolution.

COMPLAINT PORTAL OVERVIEW

The IOCL complaint portal is a user-friendly web-based system designed to allow employees and residents to register their complaints seamlessly.

Key Features of the Portal:

- Easy Registration: Users can easily log in and register new complaints.

- Categorization: The portal allows users to select from multiple complaint categories (e.g., Civil, Electrical, Furniture, Computer, Canteen, Telephone) and relevant sub-categories (e.g., Almirah, Anti-Termite, BHF, Book Shelf, Call Bell, Chimney, Fan, Geyser, etc.) ensuring precise complaint logging.

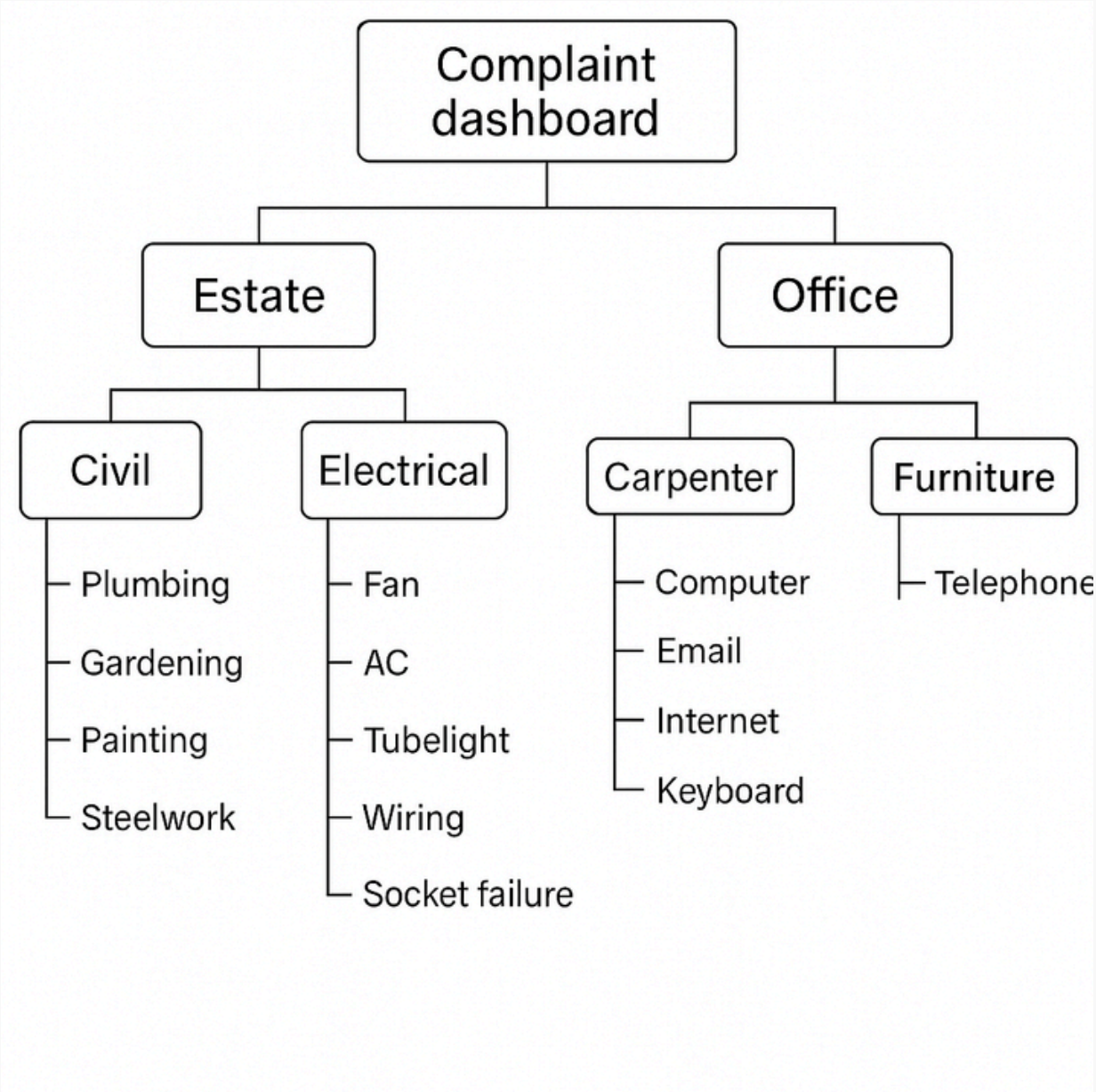
- Detailed Description: Users can provide detailed descriptions of their issues (e.g., "Wash basin tap passing check it," "Five amp switch is damaged," "LED BULB FUSED NEED TO BE GIVEN").

- Tracking: Users can track the current status of their complaints (Registered, Job Allocated, Completed, Rejected).

TOOLS USED IN POWERBI

- **Power BI**: The primary tool used for data visualization, analysis, and dashboard creation. Power BI's interactive features allow for dynamic filtering and drilling down into the data.
- **DAX (Data Analysis Expressions)**: A formula language used in Power BI (and other Microsoft Business Intelligence tools) to create custom calculations, measures, and calculated columns. DAX functions will be crucial for calculating KPIs such as:
    - TOTAL COMPLAINT = COUNTROWS(Complaints)
    - AVERAGE ALLOCATION DAYS = AVERAGE(Complaints[ALLOCATION_DAYS]) (assuming an 'ALLOCATION_DAYS' column is derived or exists)
    - AVERAGE RESOLUTION DAYS = AVERAGE(Complaints[RESOLUTION_DAYS]) (assuming a 'RESOLUTION_DAYS' column is derived or exists)
    - COMPLETED COMPLAINT = CALCULATE(COUNTROWS(Complaints), Complaints[CURRENT_STATUS_TEXT] = "Completed")
    - COMPLAINT BY CATEGORY = CALCULATE(COUNTROWS(Complaints), GROUPBY(Complaints, Complaints[COMPLAINT_CATEGORY]))
    - RANKX for Leaderboard calculations (e.g., RANKX(ALL(Employees[EMP_NAME]), [Total Complaints]))
    - Time intelligence functions to analyze trends by Year and Month.

FLOWCHAT OF THE PROCESS

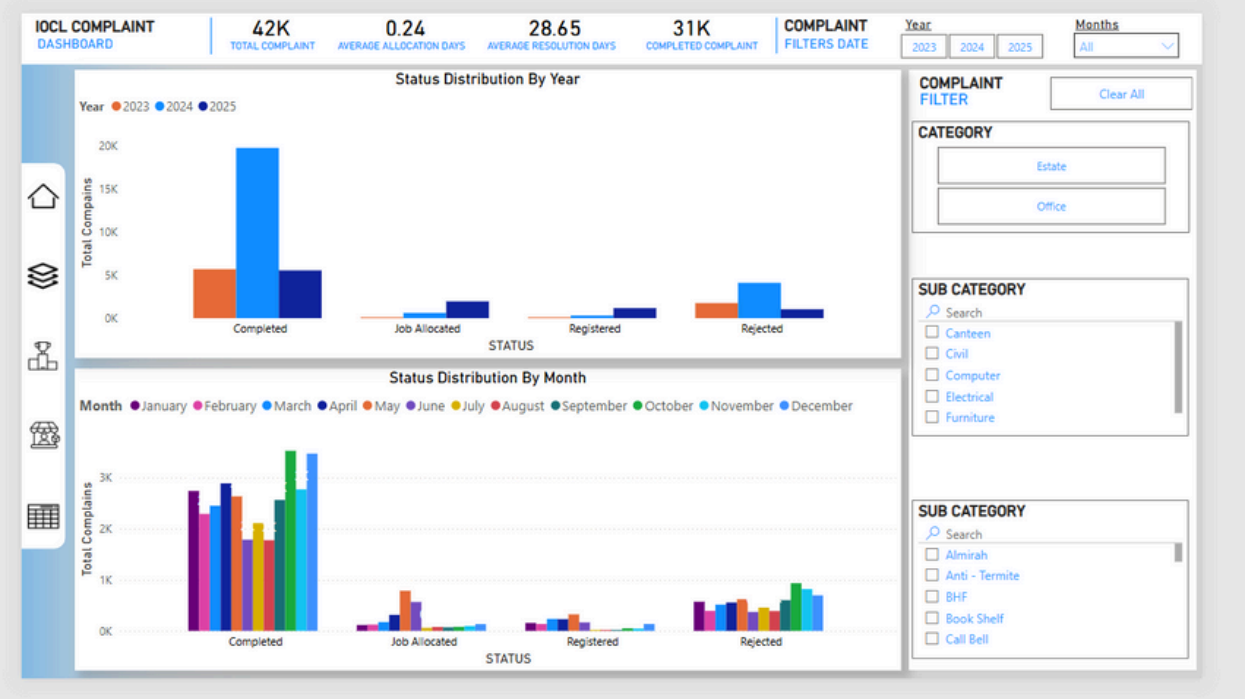


FRONT DASHBOARD

Page 1: Dashboard Overview

Purpose: To provide an executive summary of key complaint metrics, high-level trends, and overall status distribution.

- Analysis of Current Elements:
  - KPI Cards: Prominently display critical top-line metrics like Total, Average Allocation/Resolution Days, and Completed Complaints.
  - Filters: Comprehensive date (Year, Month) and detailed category/status filters are available, offering broad control.
  - Distribution Charts: "Status Distribution by Year" and "Status Distribution by Month" show complaint statuses over time weather it is completed , Rejected , job allocated , Registered
  - Category Chart: Provides a high-level breakdown of complaints by "Estate" and "Office."

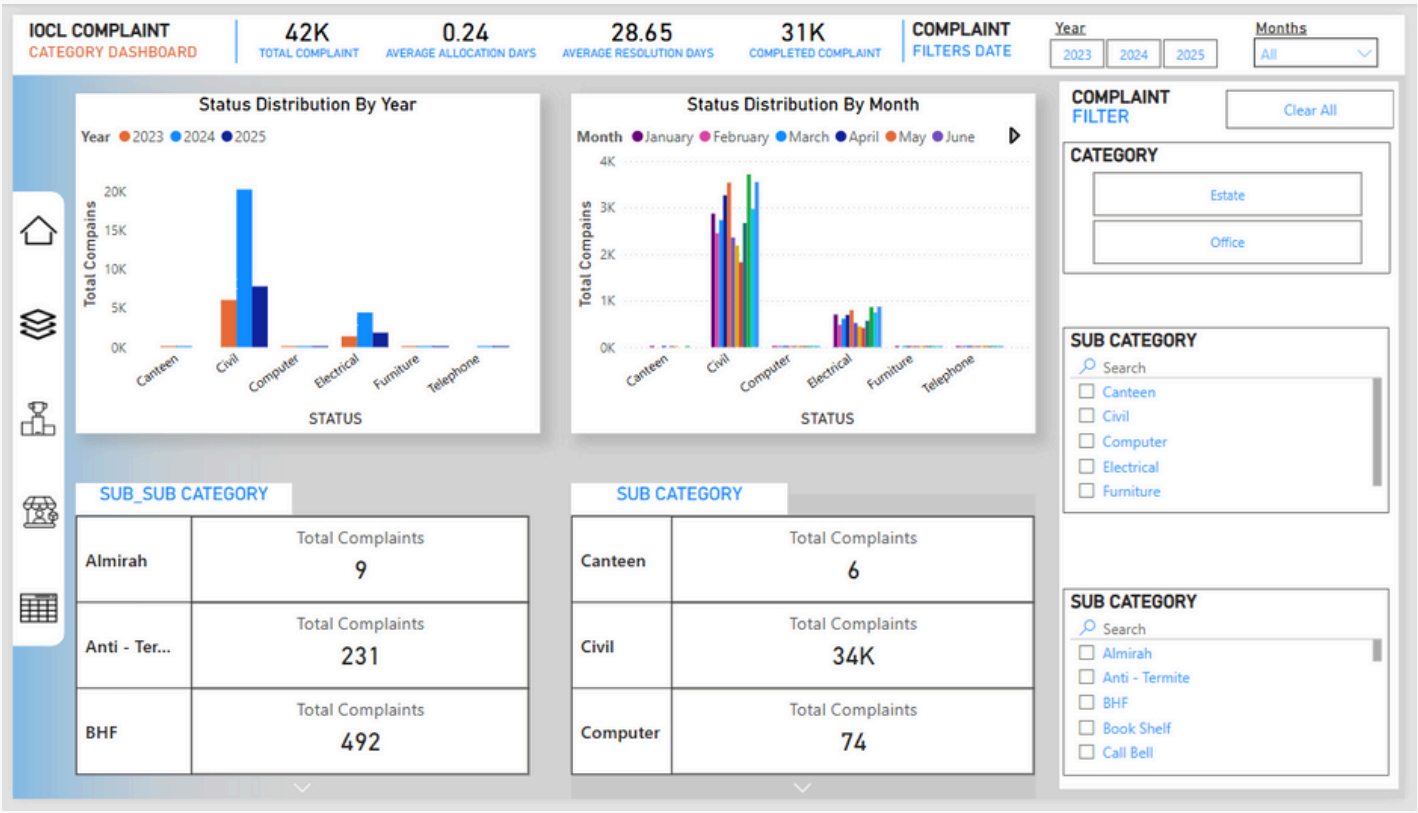


CATEGOREIES DASHBOARD

Page 2: Complaint Categories

Purpose: To offer a deeper dive into the types and classifications of complaints, identifying trends within specific categories and sub-categories.

- Analysis of Current Elements:
  - This page focuses on detailed breakdowns like "Total Complaints by COMPLAINT_SUB_CATEGORY" charts, likely featuring comparisons by status or month.
  - The presence of heavily dominant categories (e.g., "Civil" with 34K complaints) makes visual comparison with much smaller categories challenging on the same chart scale.

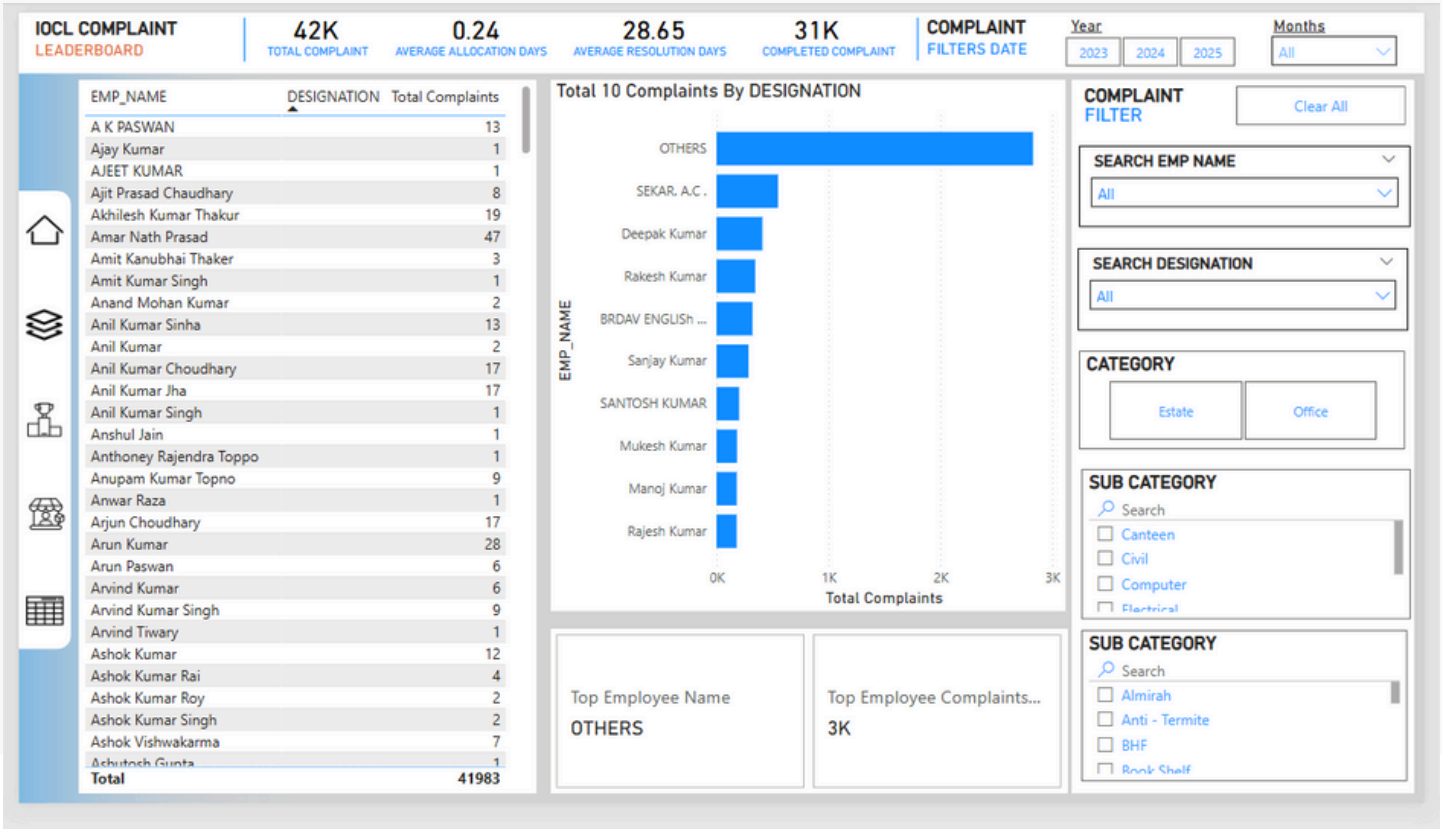


LEADERBOARD DASHBOARD

Page 3: Employee Leaderboard

Purpose: To track and assess individual employee performance in handling complaints, identifying top contributors and areas for support.

- Analysis of Current Elements:
  - Employee Table: Displays EMP_NAME, DESIGNATION, and Total Complaints, providing a granular list of employee activities.
  - Search Filters: The inclusion of search boxes for "EMP_NAME" and "DESIGNATION" is highly effective and necessary for large employee datasets, enabling quick lookup.
  - KPIs: "Top Employee Name" and "Top Employee Complaints Count" effectively highlight peak performance.
  - Bar Chart: "Total Complaints by DESIGNATION" provides an overview of workload distribution across different roles.

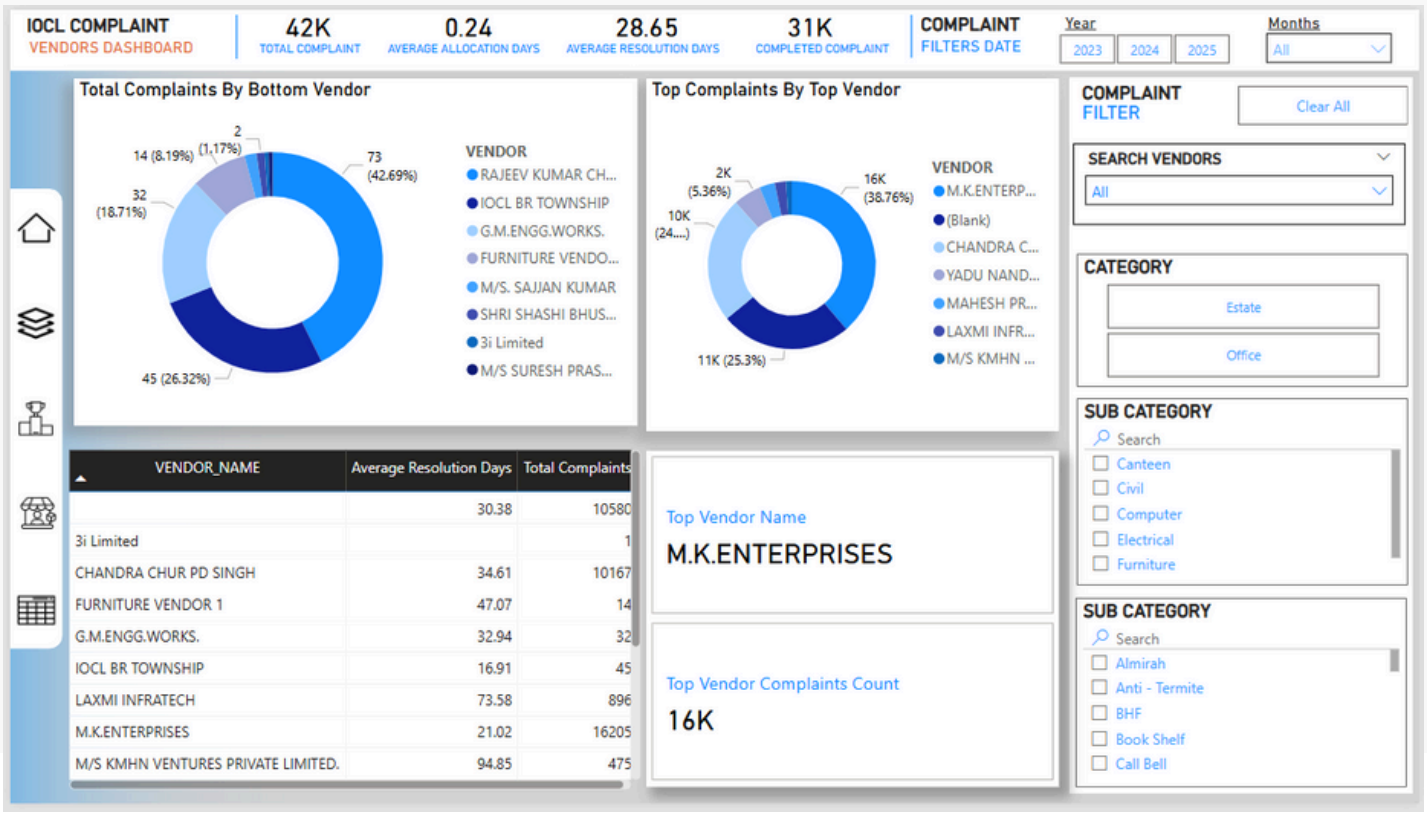


VENDOR DASHBOARD

Page 4: Vendor Performance

Purpose: To evaluate the performance of external vendors in their involvement with complaint resolution.

- Analysis of Current Elements:
  - KPI Cards: Consistent top-level KPIs are presented.
  - Vendor Donut Charts: "Total Complaints By Bottom Vendor" and "Top Complaints By Top Vendor" provide a quick visual for dominant/least common vendors.
  - Vendor Search: A search filter for VENDOR_NAME is available.
  - Vendor Table: Provides detailed metrics like VENDOR_NAME, Average Resolution Days, and Total Complaints.

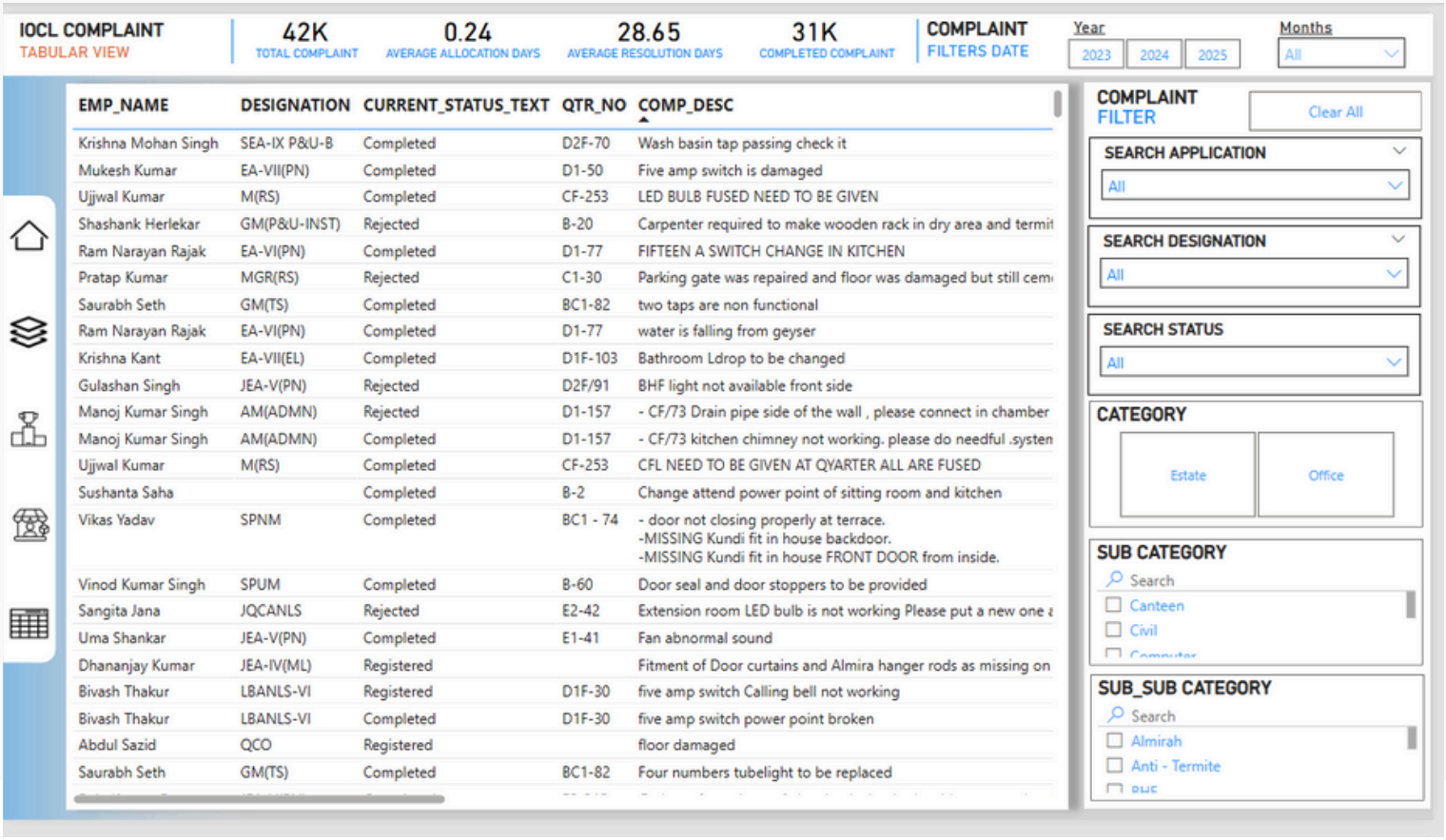


TABULAR DASHBOARD

Page 5: Tabular View

Purpose: To provide a granular, detailed breakdown of individual complaints, allowing for specific lookups and deep dives into complaint descriptions and statuses.

- Analysis of Current Elements:
  - Large Table: Displays extensive complaint details for lookup into more
  - Detailed Filters: Comprehensive search and filter options for more lookup into indepth detail about the tabular views



LEARNING AND TAKEAWAY

HSE PORTAL PROJECT

- **Importance of Regulatory Compliance**: Building an e-signature application, especially in a critical domain like HSE, highlights the absolute necessity of adhering to legal and industry-specific regulations regarding electronic signatures and document retention. This requires in-depth legal consultation and careful implementation of security measures.
- **User Experience is Paramount:** While security and compliance are non-negotiable, a clunky user experience can hinder adoption. Intuitive document uploading, clear workflow visualization, and a straightforward signing process are crucial.
- **Robust Error Handling and Logging**: Given the sensitive nature of HSE documents, comprehensive error handling and detailed logging are vital for auditing, troubleshooting, and maintaining data integrity.
- **Integration Complexities**: Integrating with existing enterprise systems (e.g., document management systems, HR platforms) or third-party e-signature providers can introduce significant integration challenges requiring careful API design and testing.
- **Change Management**: Introducing a new digital signing process requires effective change management within the organization to ensure users understand the benefits and adopt the new system. Training and clear communication are key.

LEARNING AND TAKEAWAY

POWERBI COMPLAINT DASHBOARD

- **Data Quality and Preprocessing:** The success of any dashboard heavily relies on clean, well-structured data. Issues like inconsistent naming, missing values, or incorrect data types in the raw CSV can significantly impact analysis and require robust data cleaning and transformation steps in Power BI (using Power Query).

- **Meaningful KPIs are Crucial:** Identifying and defining relevant KPIs (like average allocation/resolution days) is critical for extracting actionable insights. These metrics guide decision-making.

- **Dashboard Design for User Experience:** The arrangement of visuals, use of filters, and clear labeling are vital for user adoption and understanding. Grouping related information onto separate pages (Executive Summary, Category Analysis, Leaderboards, Vendor Performance) enhances navigation and focus.

- **Iterative Development with Stakeholder Feedback:** Dashboard development is an iterative process. Presenting drafts and gathering feedback from the Estate Office team would be essential to ensure the dashboard meets their specific needs and addresses their pain points.

- **Scalability and Performance:** As data grows, optimizing the Power BI model and DAX measures for performance becomes important. This includes efficient data loading, appropriate data types, and optimized DAX formulas.

- **Visual Storytelling**: Beyond just presenting numbers, the dashboard needs to tell a story about the complaint management process, highlighting successes and areas for improvement at a glance.

CONCLUSION

Both the **E-sign Application for HSE and the Power BI Complaint Management Dashboard projects** have offered invaluable learning experiences, highlighting critical aspects of **software development and data analytics in a corporate environment.**

The **E-sign Application for HSE** underscored the paramount importance of regulatory compliance and robust security when handling sensitive data and processes. It reinforced that user-centric design, comprehensive error handling, and effective change management are equally vital for successful adoption of digital transformation initiatives. The complexities of integrating with existing systems also provided significant insights into **API design and robust testing.**

The **Power BI Complaint Management Dashboard** project emphasized that the foundation of powerful analytics lies in high-quality, preprocessed data. It highlighted the necessity of defining meaningful KPIs to drive actionable insights and the art of dashboard design for optimal user experience. The iterative nature of dashboard development, coupled with continuous stakeholder feedback, proved crucial for delivering a solution that truly addresses business needs and facilitates visual storytelling through data. Furthermore, understanding the need for scalability and performance optimization in **data-intensive applications** was a key takeaway.

Collectively, these projects demonstrate the synergistic power of digital tools in enhancing operational efficiency, ensuring compliance, and fostering data-driven decision-making within an organization. They underscore the importance of technical skills combined with a deep understanding of functional requirements and user needs to deliver impactful solutions.