CMDA Plan Permit Scraper - Technical Documentation

Index

---

1. PROJECT OVERVIEW & ARCHITECTURE

1.1 Project Purpose

The CMDA Plan Permit Scraper is an enterprise-grade automation system designed to:

- **Scrape** planning permission documents from Chennai Metropolitan Development Authority (CMDA) website
- **Extract** structured data from PDF documents (permit details, applicant information, architect details)
- **Process** and categorize data based on business rules
- **Integrate** with Zoho CRM for lead generation and management
- **Generate** comprehensive reports and analytics
- **Automate** sales team assignment based on geographical areas

2. CORE FUNCTIONALITY BREAKDOWN

2.1 Main Functional Modules

2.1.1 PDF Data Extraction (extractor.py)

Key Components:

1. `extract_text_from_pdf_bytesio()` - Extract raw text from PDFs
2. `normalize ()` - Clean and standardize extracted text
3. `smart_split_applicant_block()` - Separate name and address intelligently
4. `extract_fields()` - Parse structured fields using regex patterns
5. `extract_area_name()` - Extract geographical area names from addresses

Technical Details:

- Uses pdfplumber for reliable text extraction (better than PyPDF2 for complex layouts)
- Implements robust regex patterns for field identification
- Handles Unicode characters and various text encodings
- Includes fallback mechanisms for incomplete data

2.1.2 Architect Information Extraction (approved_letter.py)

Purpose: Extract registered architect details from approval letters

def extract_registered_architect_from_bytes():

Searches for "Registered Architect" pattern

Extracts name, address, email, mobile from surrounding context

Uses contextual analysis (2 lines before, 7 lines after match)

2.1.3 Web Scraping Engine

- **Primary Scraper**: Playwright-based for main CMDA portal
- **Secondary Scraper**: Selenium-based for Zoho OAuth automation
- **Features**:
    - Headless browser operation
    - Dynamic content handling
    - Pagination support (10/25/50/All entries)
    - Parallel PDF downloading

2.1.4 Data Processing Pipeline

Processing Flow:

1. Extract raw text → 2. Normalize text → 3. Parse structured fields →

4. Clean and validate → 5. Enrich with area info → 6. Assign sales teams →

7. Export to Excel → 8. Push to CRM

---

3. TECHNICAL STACK & DEPENDENCIES

3.1 Core Python Libraries

Web Scraping & Automation:

playwright Modern browser automation

selenium Legacy browser automation

requests HTTP requests for PDF downloads

PDF Processing:

PyPDF2 Basic PDF operations

pdfplumber Advanced PDF text extraction

reportlab PDF report generation

Data Processing:

pandas Data manipulation and Excel operations

openpyxl Excel file creation and manipulation

re Regular expressions for text parsing

GUI Development:

PyQt5 Desktop application framework

Email & Notifications:

smtplib, email Email sending capabilities

python-dotenv Environment variable management

CRM Integration:

requests Zoho CRM API communication

3.2 System Requirements

• Python 3.8+

• Chrome/Chromium browser

• 4GB+ RAM (8GB recommended)

• Stable internet connection

• Zoho CRM account with API access

---

4. PDF PROCESSING & DATA EXTRACTION

4.1 Text Extraction Strategy

4.1.1 Multi-Library Approach

```
 Primary: pdfplumber (better layout preservation)
with pdfplumber.open(pdf_bytesio) as pdf:
    text = pdf.pages[0].extract_text()


 Secondary: PyPDF2 (for architect extraction)
reader = PyPDF2.PdfReader(pdf_bytesio)
text = page.extract_text()
```

4.1.2 Text Normalization Pipeline

```
def normalize(text):
     1. Replace special Unicode characters
    replacements = [
        ('\xa0', ' '),        Non-breaking space
        ('\u2013', '-'),      En dash
        ('\u2014', '-'),      Em dash
        ('\u201c', '"'),      Left double quote
        ('\u201d', '"'),      Right double quote
    ]
     2. Remove excessive whitespace
    text = re.sub(r'\n+', ' ', text)
    text = re.sub(r'\s+', ' ', text)


     3. Standardize formatting
    return text.strip()
```

4.2 Field Extraction Patterns

4.2.1 Regex Pattern Design

python

Example: File Number extraction

```
patterns = {
   "File No.": r"File\s*No\.?\s*[:\-]?\s*(CMDA[^\s]+)",
   "Planning Permission No.": r"Planning\s*Permission\s*No\.?\s*[:\-]?\s*([A-Z0-9/\-]+)",
   "Mobile No.": r"Mobile\s*No\.?\s*[:\-]?\s*(\d{10})",
   "Email ID": r"Email\s*ID\s*[:\-]?\s*([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})",
}
```

4.2.2 Intelligent Field Parsing

```
def smart_split_applicant_block(full_block):
   """
   Intelligently splits applicant name and address based on:
   1. Address keywords detection (Door No, Street, Nagar, etc.)
   2. Pincode patterns (6-digit numbers)
   3. Comma separation analysis
   4. Word count heuristics
   """
   address_keywords = [
      "Door No", "Old No", "New No", "Plot No", "Flat No",
      "Street", "Salai", "Nagar", "Colony", "Village",
      "Complex", "Avenue", r"[0-9]{6}"
   ]
    ... intelligent splitting logic
```

4.3 Area Name Extraction

```
def extract_area_name(site_address):
```

"""

Extracts area name from site address using hierarchical matching:

1. Village pattern: "([Area]) Village"

2. Taluk pattern: "([Area]) Taluk"

3. Chennai locality pattern: ", ([Area]), Chennai"

4. Pincode proximity pattern

"""

village_match = re.search(r"\b([A-Za-z.\s-]+?)\s+Village\b", site_address, re.IGNORECASE)

taluk_match = re.search(r"\b([A-Za-z.\s-]+?)\s+Taluk\b", site_address, re.IGNORECASE)

... additional patterns

---

5. WEB SCRAPING ENGINE

5.1 Playwright Scraping Implementation

5.1.1 Browser Configuration

```
def setup_playwright_path():
    Handles packaged executable paths for PyInstaller
    if getattr(sys, 'frozen', False):
        base_path = getattr(sys, '_MEIPASS', os.path.dirname(sys.executable))
        browser_path = os.path.join(base_path, 'ms-playwright')
        os.environ['PLAYWRIGHT_BROWSERS_PATH'] = browser_path
```

5.1.2 Scraping Workflow

```
class ScrapeWorker(QThread):
    def run(self):
        1. Launch headless browser
        playwright_instance = sync_playwright().start()
        browser = playwright_instance.chromium.launch(headless=True)

        2. Navigate to CMDA portal
        page = browser.new_page()
        page.goto(url, timeout=60000)
```

```
3. Configure table view (entries per page)
page.select_option('select[name="DataTables_Table_0_length"]', value=entry_value)

4. Extract all PDF links
links = page.locator('table tbody tr td:nth-child(9) a')
approved_links = page.locator('table tbody tr td:nth-child(7) a')

5. Download and process PDFs
for i in range(total):
    href = links.nth(i).get_attribute("href")
    full_url = urljoin("https://cmdachennai.gov.in/", href)
    ... download and process
```

5.1.3 Error-Resilient Scraping

Features implemented:

1. Timeout handling with retries
2. Progress tracking with Qt signals
3. Failed URL logging and recovery
4. Memory-efficient PDF streaming
5. Concurrent processing (limited by GUI thread)

5.2 Selenium for Zoho OAuth

5.2.1 OAuth 2.0 Automation

```
class ZohoCRMAutomatedAuth:
  def automate_oauth_flow(self, headless=False):
      1. Navigate to authorization URL
      driver.get(auth_url)

      2. Handle login form
      email_element = self.wait_and_find_element(driver, email_selectors)
      self.safe_send_keys(driver, email_element, self.email)
```

```
        3. Submit credentials
        signin_element = self.wait_and_find_element(driver, signin_selectors)
        self.safe_click(driver, signin_element)

        4. Handle 2FA banners if present
        self.handle_tfa_banner_page(driver)

        5. Extract authorization code
        if "code=" in driver.current_url:
            parsed_url = urlparse(current_url)
            code = parse_qs(parsed_url.query).get('code', [None])[0]
```

5.2.2 Robust Element Interaction

```
def safe_click(self, driver, element, description="element"):
    """Multiple click strategies with fallbacks"""
    try:
        1. Standard click
        element.click()
    except ElementNotInteractableException:
        try:
            2. JavaScript click
            driver.execute_script("arguments[0].click();", element)
        except Exception as e:
            3. Action Chains click
            ActionChains(driver).move_to_element(element).click().perform()

def safe_send_keys(self, driver, element, text, description="field"):
    """Multiple text input strategies"""
    try:
```

```
    element.clear()
    element.send_keys(text)
  except Exception as e:
     JavaScript-based input
    driver.execute_script("arguments[0].value = arguments[1];", element, text)
```

---

6. EXCEL PROCESSING & DATA TRANSFORMATION

6.1 Excel Export System

6.1.1 Structured Excel Creation

```
def export_to_xlsx(data_list, year, urls, approved_links, approved_letter, architect_details):
   Field organization
  FIELD_ORDER = [
    "File No.", "Planning Permission No.", "Permit No.",
    "Date of permit", "Date of Application",
    "Mobile No.", "Email ID",
    "Applicant Name", "Applicant Address",
    "Nature of Development", "Dwelling Unit Info",
    "Site Address", "Area Name"
  ]

   Additional columns
  headers = FIELD_ORDER + [
    "Architect Name", "Architect Address",
    "Architect Email", "Architect Mobile",
    "View Online", "Approved Plan", "Approval Letter"
  ]

   Hyperlink integration
  link_cell = ws.cell(row=row_idx, column=len(FIELD_ORDER) + 5, value="View PDF")
  if url:
    link_cell.hyperlink = url
```

```
    link_cell.style = "Hyperlink"
```

6.1.2 Dual File Export Strategy

```
 1. Temporary file (for CRM processing)
temp_dir = Path(tempfile.gettempdir())
temp_path = temp_dir / f"CMDA_{year}.xlsx"


 2. User download file
downloads_folder = Path.home() / "Downloads"
download_path = downloads_folder / f"CMDA_{year}.xlsx"
```

6.2 Data Comparison & Update System

6.2.1 Incremental Processing

```
def compare_and_update_excel(new_file):
   """
   Compares new scraped data with existing database:
   1. Load existing data from "ExistData.xlsx"
   2. Filter out invalid entries (Failed, Error, Not Found)
   3. Identify new records not in existing database
   4. Merge new records with existing data
   5. Save updated database and return new records
   """
    Key comparison on "Planning Permission No."
   new_entries = valid_new_df[~valid_new_df[key_col].isin(exist_df[key_col])]


    Merge and save
   updated_exist_df = pd.concat([exist_df, new_entries], ignore_index=True)
```

6.2.2 No-Records Alert System

```
def send_no_new_records_alert():
   """Sends email notification when no new records found"""
   if valid_new_df.empty:
      send_no_new_records_alert()
      return False, {}
```

7. ZOHO CRM INTEGRATION

7.1 Authentication & Token Management

7.1.1 OAuth 2.0 Token Flow

```
class ZohoCRMAutomatedAuth:
    def ensure_valid_token(self):
        1. Check if token exists
        if not self.access_token:
            self.load_tokens()

        2. Check token expiration
        if self.token_expires_at and datetime.now() >= self.token_expires_at:
            if not self.refresh_access_token():
                3. Full re-authentication if refresh fails
                return self.automate_oauth_flow()

        4. Initial authentication if no token
        if not self.access_token:
            return self.automate_oauth_flow()

        return True
```

7.1.2 Token Persistence

```
def save_tokens(self):
    token_data = {
        'access_token': self.access_token,
        'refresh_token': self.refresh_token,
        'expires_at': self.token_expires_at.isoformat() if self.token_expires_at else None,
        'client_id': self.client_id
    }
    with open(self.token_file, 'w') as f:
        json.dump(token_data, f, indent=2)
```

7.2 Data Mapping & Transformation

7.2.1 Field Mapping Strategy

```
field_mapping = {
   "Sales Person": "Lead_Owner",
   "Email ID": "Email",
   "Mobile No.": "Mobile_Number",
   "Date of permit": "Date_of_Permit",
   "Applicant Name": "Lead_Name",
   "Nature of Development": "Nature_of_Developments",
   "Dwelling Unit Info": "Dwelling_Unit_Info",
    ... additional mappings
}
```

7.2.2 Intelligent Data Transformation

```
def format_record_for_zoho(self, record):
    1. Calculate bathrooms from dwelling units
   dwelling_units = record.get("Dwelling Unit Info")
   if dwelling_units:
      numbers = re.findall(r'\d+', str(dwelling_units))
      if numbers:
         dwelling_value = int(numbers[0])
         bathrooms = dwelling_value * 2
         formatted_record["No_of_bathrooms"] = str(bathrooms)

    2. Date formatting
   if excel_field in ["Creation_Time", "Date_of_Permit"]:
      formatted_record[zoho_field] = dt.strftime("%Y-%m-%dT%H:%M:%S+05:30")

    3. Email validation
   elif excel_field == "Email ID":
      if "@" in email_str and "." in email_str:
         formatted_record[zoho_field] = email_str
```

7.3 Lead Creation & Management

7.3.1 Lead Creation Pipeline

```
def create_lead_from_cmda_record(self, cmda_record):
    1. Prepare lead data
   lead_data = {
      "Planning_Permission_No": self.clean_value(cmda_record.get("Planning Permission No.", "")),
      "Email": self.clean_value(cmda_record.get("Email ID", "")),
      "Phone": self.clean_value(cmda_record.get("Mobile No.", "")),
       ... additional fields
   }


    2. Handle sales person assignment
   sales_person = cmda_record.get("Sales Person", "")
   if sales_person:
      owner_id = self.get_user_id_by_name(sales_person)
      if owner_id:
         lead_data["Owner"] = owner_id


    3. API request to Zoho
   url = f"{self.api_base_url}/Leads"
   response = requests.post(url, json={'data': [lead_data]}, headers=headers)
```

7.3.2 Batch Processing

```
def push_records_to_zoho(self, records, batch_size=100):
   """Process records in batches to avoid API limits"""
   for i in range(0, total_records, batch_size):
      batch = records[i:i + batch_size]
      formatted_batch = []


      for record in batch:
         formatted_record = self.format_record_for_zoho(record)
```

```
        formatted_batch.append(formatted_record)


    API call with batch
    payload = {'data': formatted_batch}
    response = requests.post(url, json=payload, headers=headers)
```

---

8. EMAIL NOTIFICATION SYSTEM

8.1 Alert Categories

8.1.1 Unmatched Areas Alert

```
def send_unmatched_areas_alert(unmatched_df, original_file_name):
    """
    Sends alert when areas cannot be matched to salespersons
    Includes:
    1. Summary statistics
    2. List of unmatched areas
    3. Attachment with detailed data
    4. Action recommendations
    """
    total_unmatched = len(unmatched_df)
    unique_areas = unmatched_df['Area Name'].nunique()


    HTML email with styling
    body = f'''
    <html>
    <body style="font-family: Arial, sans-serif;">
        <h2> ⚠️ Unmatched Areas Alert</h2>
        <p>Total Unmatched Records: {total_unmatched}</p>
        <p>Unique Unmatched Areas: {unique_areas}</p>
        <!-- Additional content -->
    </body>
    </html>
```

'''

8.1.2 Records Processing Alert

```
def send_records_alert(matched_df, unmatched_df, original_file_name):
    """
    Comprehensive report of matched vs unmatched records
    Includes detailed statistics and attachments
    """
    total_matched = len(matched_df)
    total_unmatched = len(unmatched_df)
    total_records = total_matched + total_unmatched


     Visual dashboard in email
    body = f'''
    <div style="display: flex; gap: 15px;">
      <div style="background-color: d4edda;">
        <h4> ✅ Matched Records</h4>
        <p>{total_matched}</p>
      </div>
      <div style="background-color: f8d7da;">
        <h4> ⚠️ Unmatched Records</h4>
        <p>{total_unmatched}</p>
      </div>
    </div>
    '''
```

8.2 Email Configuration & Security

8.2.1 SMTP Configuration

```
def send_email(self, body, attachments=[]):
     Gmail SMTP configuration
    with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
      server.login(sender_mailId, passKey)
      server.sendmail(sender_mailId, recipient_email, msg.as_string())
```

8.2.2 Attachment Handling

```
 Create temporary Excel files
temp_file = tempfile.NamedTemporaryFile(delete=False, suffix=".xlsx")
unmatched_df.to_excel(temp_file.name, index=False)


 Attach to email
with open(temp_file_path, 'rb') as f:
   attachment = MIMEApplication(f.read(),
      _subtype='vnd.openxmlformats-officedocument.spreadsheetml.sheet')
   attachment.add_header('Content-Disposition',
      'attachment', filename=attachment_filename)
   msg.attach(attachment)
```

---

9. GUI APPLICATION (PYQT5)

9.1 Application Architecture

9.1.1 Main Window Structure

```
class ScraperApp(QWidget):
   def __init__(self):
      super().__init__()
      self.setWindowTitle("CMDA Plan Permit Scraper")
      self.setMinimumSize(1200, 800)
      self.setup_ui()

   def setup_ui(self):
       1. Header with logo and title
      header_layout = QVBoxLayout()

       2. Year selection group
      year_group = QGroupBox("Choose Year")
```

```
        3. Entry count selection
        self.entry_group = QGroupBox("Select Entry Count")

        4. Progress indicators
        self.progress = QProgressBar()
        self.loader = QLabel()

        5. Action buttons
        self.scrape_btn = QPushButton("Start Scraping")
        self.report_btn = QPushButton("Generate PDF Report")
```

9.1.2 Styling & Theming

```
 Professional styling with company colors
self.setStyleSheet("""
    QWidget {
        font-family: 'Segoe UI', 'Inter', sans-serif;
        background-color: ffffff;
    }

    QLabelTitleLabel {
        color: dc2626;  /* Company red */
        font-size: 36px;
        font-weight: 800;
    }

    QPushButton {
        background-color: dc2626;
        color: white;
        border-radius: 8px;
        padding: 10px 24px;
    }
```

```
  QPushButton:hover {
    background-color: b91c1c;
  }
""")
```

9.2 Threading & Concurrency

9.2.1 Background Worker Implementation

```
class ScrapeWorker(QThread):
  progress = pyqtSignal(int, int)      Progress updates
  finished = pyqtSignal(list, tuple)   Completion signal
  error = pyqtSignal(str)              Error signal

  def run(self):
      Long-running scraping operation
      Emits signals for GUI updates
     self.progress.emit(current, total)
```

9.2.2 Thread-Safe GUI Updates

```
def update_progress(self, current, total):
   Called from worker thread via signal
  self.progress.setMaximum(total)
  self.progress.setValue(current)
  QApplication.processEvents()   Keep GUI responsive
```

9.3 User Experience Features

9.3.1 Real-Time Feedback

```
def load_year(self, year):
   Show loading animation
  self.loader.setVisible(True)
  self.loader_movie.start()

   Fetch row count asynchronously
  success = self.update_row_count()
```

```
    Update UI based on result
    self.entry_group.setVisible(success)
    self.scrape_btn.setVisible(success)
```

9.3.2 Completion Dialog

```
def show_completion_message(self, pdf_count, import_result, download_path):
    Rich HTML dialog with statistics
    message = f"""
    <div style='font-family: Segoe UI;'>
        <h3 style='color: dc2626;'> ✅ SCRAPING COMPLETED</h3>
        <p>Documents Processed: {pdf_count}</p>
        <p>File Saved: {download_path}</p>
    </div>
    """

    msg_box = QMessageBox()
    msg_box.setTextFormat(Qt.RichText)
    msg_box.setText(message)
    msg_box.exec_()
```

---

10. REPORT GENERATION SYSTEM

10.1 PDF Report Generation

10.1.1 ReportLab Implementation

```
def generate_pdf_report(file_path, scraping_stats, crm_result, year, local_file_path):
    doc = SimpleDocTemplate(
        file_path,
        pagesize=A4,
        rightMargin=72,
        leftMargin=72,
        topMargin=72,
        bottomMargin=72
```

```
    )

    elements = []

     1. Title and header
    elements.append(Paragraph("CMDA SCRAPING REPORT", title_style))

     2. Scraping statistics table
    scraping_data = [
        ["Metric", "Count"],
        ["Total Records Attempted", str(scraping_stats.get('total_attempted', 0))],
        ["Successfully Scraped", str(scraping_stats.get('successful_scraped', 0))],
        ["Failed to Scrape", str(scraping_stats.get('failed_scraped', 0))],
    ]

     3. CRM integration results
     4. Data analysis section
     5. Footer
```

10.1.2 Professional Styling

```
 Custom styles for professional appearance
title_style = ParagraphStyle(
    'CustomTitle',
    fontSize=24,
    alignment=TA_CENTER,
    textColor=colors.HexColor('dc2626'),
    spaceAfter=20
)

 Table styling
```

```
scraping_table.setStyle(TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('dc2626')),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('GRID', (0, 0), (-1, -1), 1, colors.black),
]))
```

10.2 Report Content Sections

10.2.1 Scraping Statistics

- Total records attempted
- Successfully scraped count
- Failed records with file numbers
- Success rate percentage

10.2.2 CRM Integration Results

- Import status (Success/Failed)
- Error messages if any
- Records pushed to CRM

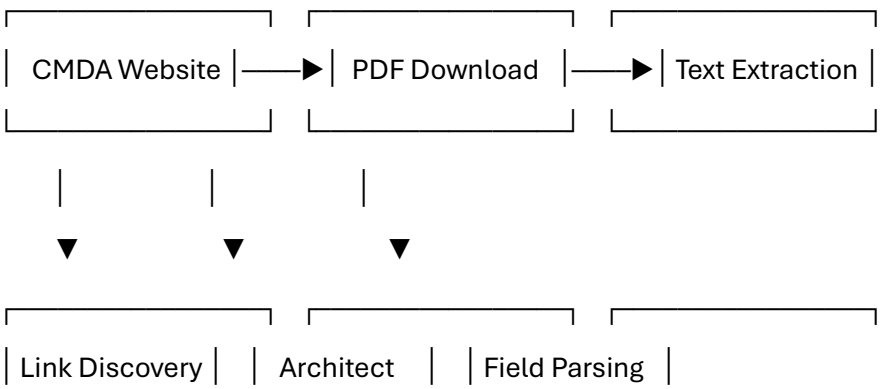
10.2.3 Data Analysis

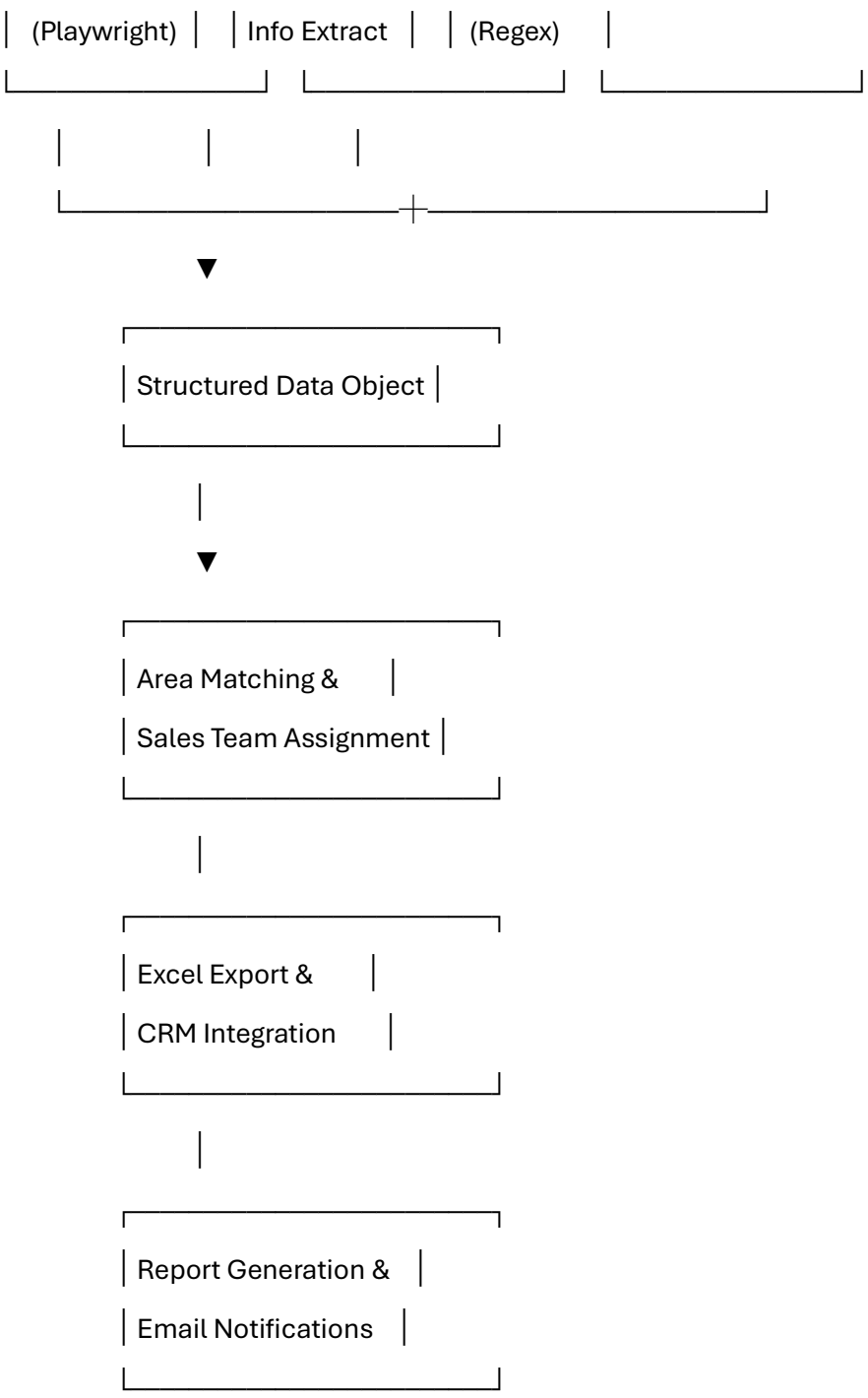
- New records count
- Matched vs unmatched records
- Unmatched areas list
- File number listings

---

11. DATA FLOW ARCHITECTURE

11.1 End-to-End Processing Pipeline





11.2 Data Validation & Quality Control

11.2.1 Validation Checkpoints

1. PDF Download Validation

```
if r.status_code != 200:
   mark_as_failed(file_no_text)
```

2. Text Extraction Validation

```
text = extract_text_from_pdf_bytesio(pdf_io)
if not text or len(text) < 50:
    mark_as_failed(file_no_text)

 3. Field Extraction Validation
if fields.get("File No.") in ["Not Found", "Error", ""]:
    mark_as_failed(file_no_text)

 4. Area Matching Validation
if not find_best_match(area_name):
    add_to_unmatched_areas(area_name)
```

11.2.2 Data Cleaning Pipeline

```
def clean_value(self, value):
    """
    Comprehensive data cleaning:
    1. Handle None/NaN values
    2. Remove whitespace
    3. Filter invalid strings
    4. Standardize formatting
    """
    if value is None or pd.isna(value):
        return ""

    value_str = str(value).strip()

    if value_str.lower() in ['', 'nan', 'none', 'null']:
        return ""

    return value_str
```

---

12. ERROR HANDLING & LOGGING

12.1 Multi-Layer Error Handling

12.1.1 Scraping Error Recovery

```
try:
    Attempt primary scraping method
   r = requests.get(full_url, timeout=30)
   r.raise_for_status()
except requests.exceptions.RequestException as e:
    Log failure and continue with other records
   self.failed_scraped += 1
   self.failed_file_numbers.append(file_no_text)


    Add placeholder record to maintain data structure
   pdf_streams.append({
      "File No.": file_no_text,
      "Planning Permission No.": "Failed",
       ... other fields marked as Failed
   })
```

12.1.2 CRM API Error Handling

```
try:
   response = requests.post(url, json=payload, headers=headers)
   if response.status_code == 201:
       Process successful response
      pass
   else:
       Log API error details
      print(f"❌ HTTP Error {response.status_code}: {response.text}")
       Implement retry logic or fallback
except Exception as e:
    Log exception with traceback
   traceback.print_exc()
```

12.2 Comprehensive Logging Strategy

12.2.1 Progress Logging

Real-time progress updates

print(f"✅ Matched data saved to: {matched_temp_file.name}")

print(f" Total matched records: {len(matched_df)}")

Error logging with context

print(f"❌ Error in separate_and_store_temp: {e}")

Success confirmation

print(f"✅ Email report sent successfully to {recipient_email}")

12.2.2 File Operation Logging

python

File save locations

print(f"📁 File saved to Downloads: {download_path}")

print(f"📁 Temporary file: {temp_path}")

Data statistics

print(f"📊 Total records: {total_records}")

print(f"✅ Successful: {successful_records}")

print(f"❌ Failed: {failed_records}")

---

13. DEPLOYMENT & CONFIGURATION

13.1 Environment Configuration

13.1.1 .env File Structure

env

Zoho CRM Configuration

CLIENT_ID=your_client_id_here

CLIENT_SECRET=your_client_secret_here

REDIRECT_URL=https://your-redirect-url.com

ORG_ID=your_organization_id

```
EMAIL_ADDRESS=your_email@company.com
PASSWORD=your_password

 API URLs
AUTH_URL=https://accounts.zoho.com/oauth/v2/auth
TOKEN_URL=https://accounts.zoho.com/oauth/v2/token
API_BASE_URL=https://www.zohoapis.com/crm/v2

 Email Configuration
SENDER_MAIL=your_email@gmail.com
APP_PASSWORD=your_app_specific_password
RECIPIENT_MAIL=recipient@company.com

 Application Settings
ZOHO_MODEL_NAME=CMDA_Data
TOKEN_FILE_NAME=zoho_tokens.json

 Sales Team User IDs
ZOHO_USER_ID_ABHISHEK=1234567890
ZOHO_USER_ID_KARTHIK=2345678901
ZOHO_USER_ID_JAGAN=3456789012
```

13.1.2 Configuration Validation

```
def validate_configuration():
    required_env_vars = [
        "CLIENT_ID", "CLIENT_SECRET", "REDIRECT_URL",
        "ORG_ID", "EMAIL_ADDRESS", "PASSWORD",
        "SENDER_MAIL", "APP_PASSWORD", "RECIPIENT_MAIL"
    ]

    missing_vars = []
    for var in required_env_vars:
```

```
        if not os.getenv(var):
            missing_vars.append(var)

    if missing_vars:
        print(f"❌ Missing environment variables: {', '.join(missing_vars)}")
        return False

    return True
```

13.2 Installation & Setup

13.2.1 Prerequisites Installation

1. Install Python dependencies

```
pip install playwright pdfplumber PyPDF2 pandas openpyxl
pip install PyQt5 reportlab python-dotenv requests
pip install selenium email
```

2. Install Playwright browsers

```
playwright install chromium
```

3. Set up environment variables

```
cp .env.example .env
```

Edit .env with your credentials

13.2.2 Executable Packaging

PyInstaller configuration for standalone executable

```
pyinstaller --onefile --windowed --add-data "client_logo.png;."
--add-data "loader.gif;." --add-data "ms-playwright;ms-playwright"
main.py
```

13.3 Performance Optimization

13.3.1 Memory Management

Stream PDF processing to avoid memory issues

```
pdf_io = BytesIO(r.content)   Keep in memory stream
text = extract_text_from_pdf_bytesio(pdf_io)
pdf_io.close()   Explicitly close stream
```

Batch processing for CRM integration

batch_size = 100 Optimal batch size for Zoho API

13.3.2 Network Optimization

Request timeout configuration

timeout=30 Balance between reliability and performance

Concurrent downloads (limited by GUI responsiveness)

Consider implementing thread pool for production use

13.4 Security Considerations

13.4.1 Credential Security

- Store sensitive data in .env file (excluded from version control)
- Use app-specific passwords for email services
- Implement token encryption for production use
- Regular credential rotation

13.4.2 Data Privacy

- Anonymize sensitive data in logs
- Secure temporary file deletion
- Implement data retention policies
- GDPR compliance considerations

CONCLUSION

This CMDA Plan Permit Scraper represents a sophisticated enterprise automation solution that combines web scraping, data processing, CRM integration, and reporting into a cohesive system. Key strengths include:

1. **Robust Architecture**: Multi-layered design with clear separation of concerns
2. **Comprehensive Error Handling**: Graceful degradation and recovery mechanisms
3. **Professional UI**: User-friendly interface with real-time feedback
4. **Scalable Design**: Batch processing and efficient memory management
5. **Extensive Integration**: Seamless connection with Zoho CRM and email systems
6. **Detailed Reporting**: Comprehensive analytics and PDF report generation

The system is production-ready with proper configuration and can be extended for additional data sources or CRM platforms as needed.