Shoalhaven DA Tracking Web Scraper - Complete Documentation

Executive Summary

This document provides comprehensive documentation for the **Shoalhaven DA Tracking Web Scraper**, a production-grade Python application designed to extract development application (DA) records from the Shoalhaven City Council website. The scraper automates the process of searching, collecting, and organizing DA information into a structured CSV format with exact data cleaning rules.

Table of Contents

1. Project Overview

Purpose

The Shoalhaven DA Tracking Web Scraper extracts development application records from the Shoalhaven City Council's online DA Tracking system for a specified date range (01/09/2025 - 30/09/2025). The scraper is designed to:

- Automate manual data collection
- Ensure consistency and accuracy
- Prevent duplicate records
- Apply standardized data cleaning rules
- Export data in a structured CSV format

Key Features

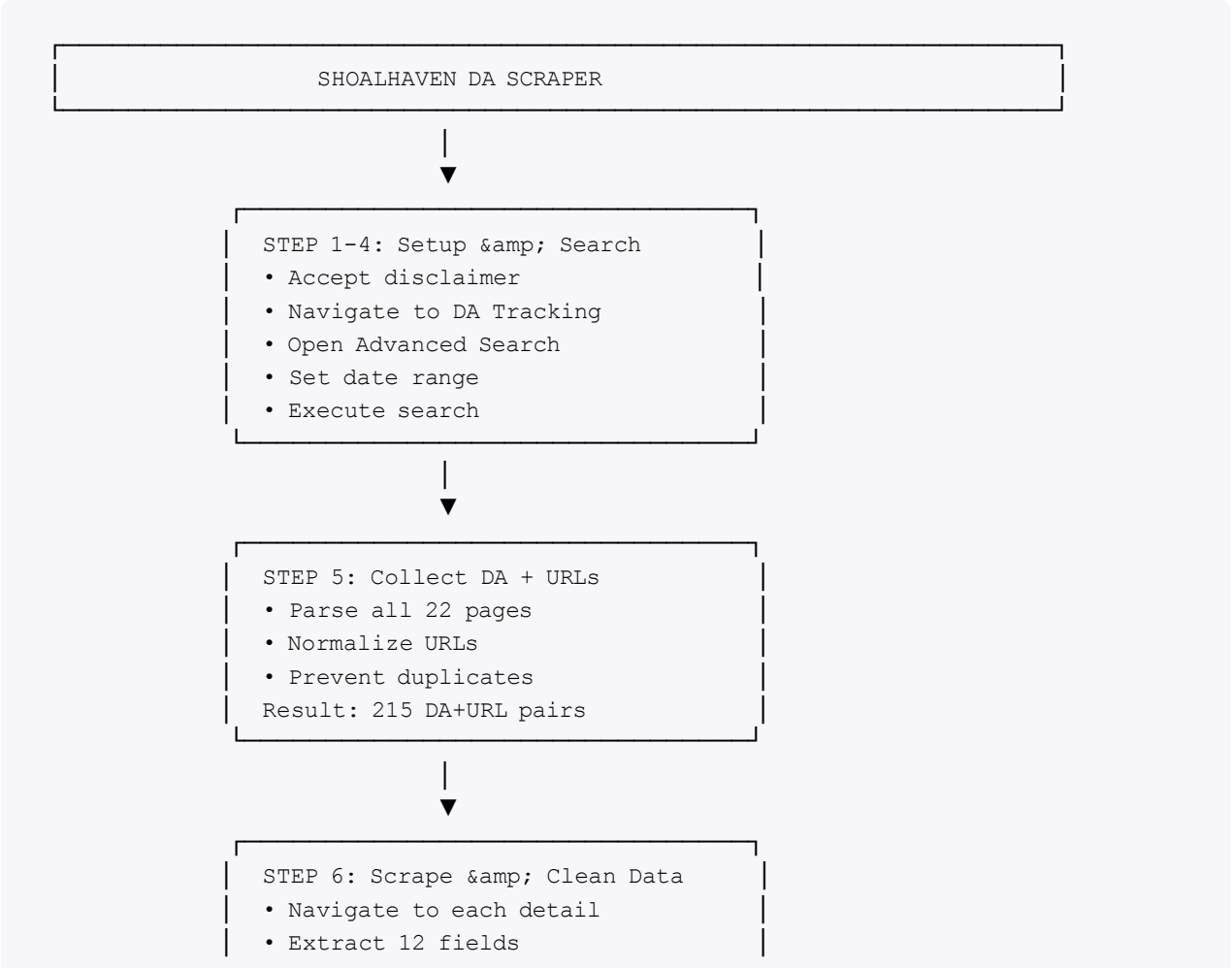
- **Automated Web Navigation**: Uses Selenium WebDriver to navigate the complex council website
- **URL Normalization**: Converts relative URLs to absolute URLs for reliable navigation
- **Duplicate Prevention**: Tracks processed records to prevent duplicates
- **Sequential Scraping**: Processes records one-by-one for reliability
- **Exact Data Cleaning**: Implements character-for-character text matching rules
- **Structured Output**: Generates a 12-column CSV with standardized headers
- **Error Recovery**: Includes robust error handling and recovery mechanisms

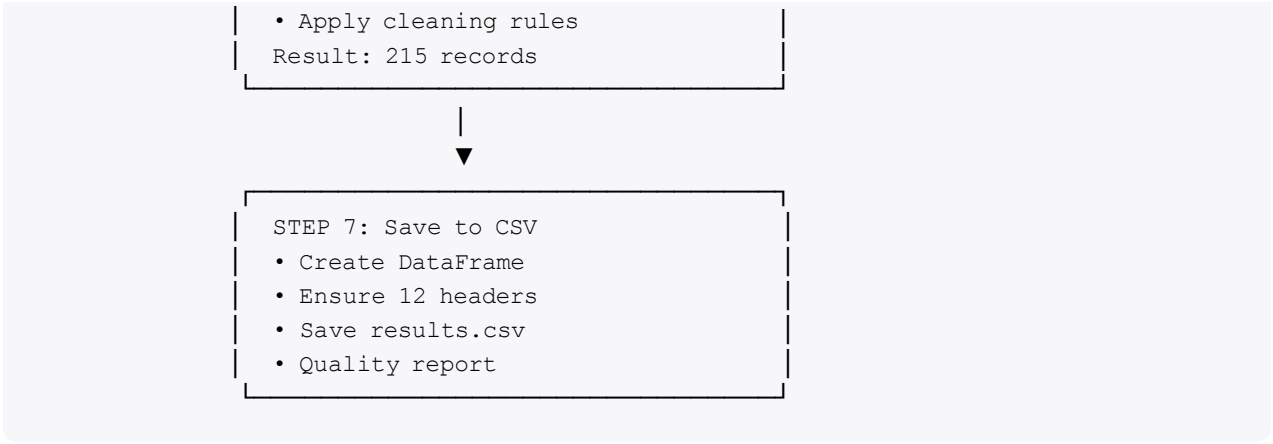
Target Audience

- Data analysts and researchers studying development applications
- Council staff automating data collection
- Developers building upon this scraper for similar websites

2. System Architecture

High-Level Workflow





Technology Stack

- **Selenium WebDriver**: Browser automation and navigation
- **BeautifulSoup**: HTML parsing and element extraction
- **Pandas**: Data manipulation and CSV export
- **Python 3.7+**: Programming language

3. Dependencies & Installation

Required Libraries

```
selenium&gt;=4.0.0
webdriver-manager&gt;=3.8.0
beautifulsoup4&gt;=4.9.0
pandas&gt;=1.3.0
```

Installation Steps

```
# 1. Create virtual environment
python -m venv scraper_env

# 2. Activate virtual environment
# On Windows:
scraper_env\Scripts\activate
# On macOS/Linux:
source scraper_env/bin/activate

# 3. Install dependencies
pip install selenium webdriver-manager beautifulsoup4 pandas

# 4. Run the scraper
python scraper.py
```

System Requirements

- Windows, macOS, or Linux
- Python 3.7 or higher
- 500 MB free disk space
- Active internet connection
- Google Chrome browser (installed locally)

4. Configuration

Configuration Variables

Located at the top of the script:

```
START_DATE = "01/09/2025"          # Search start date (DD/MM/YYYY)
END_DATE = "30/09/2025"            # Search end date (DD/MM/YYYY)
BASE_URL = "https://..."           # Council website base URL
BASE_APP = "https://.../ApplicationMaster/" # Application module base
BASE_ROOT = "https://www3.shoalhaven.nsw.gov.au"  # Site root
OUTPUT_CSV = "results.csv"         # Output CSV filename
WAIT_TIME = 20                     # Selenium wait timeout (seconds)
```

Data Cleaning Configuration

```
# Exact text to match for Fees field
FEES_NO_FEES_TEXT = "No fees recorded against this application."

# Exact text to match for Contact Council field
CONTACT_NO_EXHIBITION_TEXT = "Application Is Not on exhibition, please call Council on 13
```

CSV Header Configuration

```
HEADERS = [
    "DA_Number",           # Development application number
    "Detail_URL",          # Full URL to detail page
    "Description",         # Application description
    "Submitted_Date",      # Date application was submitted
    "Decision",            # Decision made on application
    "Categories",          # Application type/category
    "Property_Address",    # Property address
    "Applicant",          # Applicant name/organization
    "Progress",           # Current progress status
    "Fees",               # Fees (cleaned field)
    "Documents",          # Associated documents
    "Contact_Council",    # Contact information (cleaned field)
]
```

5. Function Documentation

A. Driver Initialization

`create_driver(headless=False)`

Purpose: Creates and configures a Chrome WebDriver instance for browser automation.

Parameters:

- `headless` (bool): If True, runs browser in headless mode (no visible window). Default: False.

Returns: WebDriver object configured with specific options.

Key Configurations:

- `--no-sandbox`: Disables sandbox for Docker compatibility
- `--disable-dev-shm-usage`: Prevents memory issues
- `--window-size=1920x1080`: Sets consistent viewport size
- 60-second page load timeout

Usage:

```
driver = create_driver(headless=False)  # Shows browser window
driver = create_driver(headless=True)  # Headless mode
```

B. Setup & Search Functions (Steps 1-4)

`step_1_accept_disclaimer(driver)`

Purpose: Navigates to the website and accepts the disclaimer/terms.

Process:

1. Opens the base URL
2. Waits up to 20 seconds for "Agree" button to appear
3. Clicks the "Agree" button
4. Waits 2 seconds for page to load

Returns: True if successful, True even if button not found (graceful handling).

Example Output:

```
STEP 1: Accept Disclaimer
o Clicked 'Agree' button
```

`step_2_navigate_da_tracking(driver)`

Purpose: Navigates from the home page to the DA Tracking module.

Process:

1. Waits for "DA Tracking" link to be clickable
2. Clicks the DA Tracking navigation link
3. Waits 3 seconds for module to load

Returns: True if successful, False if failed.

XPath Used: `//a[.//span[text()='DA Tracking']]`

Example Output:

```
STEP 2: Navigate to DA Tracking
o Navigated to DA Tracking
```

`step_3_open_advanced_search(driver)`

Purpose: Opens the Advanced Search panel for filtered searches.

Process:

1. Waits for "Advanced Search" link to appear
2. Clicks the link using JavaScript execution
3. Waits 2 seconds for panel to open

Returns: True if successful, False if failed.

Note: Uses `driver.execute_script()` instead of regular click for reliability with Telerik controls.

Example Output:

```
STEP 3: Open Advanced Search
o Opened Advanced Search panel
```

`step_4_set_date_range(driver, start_date, end_date)`

Purpose: Sets the search date range in the Advanced Search panel.

Parameters:

- `driver`: WebDriver instance
- `start_date` (str): Start date in format "DD/MM/YYYY" (e.g., "01/09/2025")
- `end_date` (str): End date in format "DD/MM/YYYY" (e.g., "30/09/2025")

Process:

1. Locates FROM date input field using XPath
2. Clicks field, clears existing value
3. Types date character-by-character (50ms delay between chars)
4. Presses TAB to confirm and move to next field
5. Repeats for TO date field

Returns: True if successful, False if failed.

Why Character-by-Character Input?:

- Ensures keyboard events are properly registered
- Compatible with ASP.NET date picker controls
- More reliable than paste operations

Example Output:

```
STEP 4: Set Date Range
o Date range set: 01/09/2025 → 30/09/2025
```

`step_4b_click_search(driver)`

Purpose: Clicks the Search button and waits for results to load.

Process:

1. Locates Search button by element ID
2. Scrolls button into view
3. Clicks the button using JavaScript
4. Waits for results table to appear
5. Waits additional 2 seconds for full load

Returns: True if results loaded, False if failed.

Waits For: `//table[contains(@class,'rgMasterTable')]` OR `//span[contains(text(),'No records')]`

Example Output:

```
STEP 4B: Click Search
o Clicked Search button
o Results grid loaded
```

C. Pagination & Data Extraction Functions

`extract_total_pages_and_items(driver)`

Purpose: Extracts total number of pages and items from search results.

Process:

1. Parses HTML with BeautifulSoup
2. Finds div with class matching regex `rgWrap.*rgInfoPart`
3. Extracts all `<strong>` tags from that div
4. First `<strong>` = total items (215)
5. Second `<strong>` = total pages (22)

Returns: Tuple (total_items, total_pages) or (None, None) if failed.

HTML Pattern Matched:

```
<div>
     <strong>215</strong> items in <strong>22</strong> pages
</div>
```

Example Output:

```
 Search Results Detected:
   Total Items: 215
   Total Pages: 22
```

`normalize_url(raw_url)`

Purpose: Converts relative URLs to absolute URLs for proper navigation.

Process:

1. Checks if URL already absolute (starts with http:// or https://)
   - If yes, returns as-is
2. Checks if starts with `default.aspx`
   - If yes, prepends BASE_APP: `https://www3.shoalhaven.nsw.gov.au/masterviewUI/modules/ApplicationMaster/`
3. Checks if starts with `/`
   - If yes, prepends BASE_ROOT: `https://www3.shoalhaven.nsw.gov.au`
4. Default fallback: assumes ApplicationMaster-relative

Returns: Full absolute URL.

Examples:

- `default.aspx?page=wrapper&key=732614` → `https://www3.shoalhaven.nsw.gov.au/masterviewUI/modules/ApplicationMaster/default.aspx?page=wrapper&key=732614`
- `/documents/file.pdf` → `https://www3.shoalhaven.nsw.gov.au/documents/file.pdf`
- `https://example.com` → `https://example.com` (unchanged)

`parse_row_link_href(row)`

Purpose: Extracts href attribute from Show button in table row.

Process:

1. Finds `<a>` tag in row
2. Gets href attribute
3. Skips JavaScript links (starting with "javascript:")
4. Returns href value or None

Returns: href string or None if not found.

Why Skip JavaScript Links?:

- JavaScript postbacks (`javascript:__doPostBack(...)`) cannot be used with `driver.get()`
- Only direct URLs can be navigated to

`collect_da_and_urls(driver, total_pages)`

Purpose: Collects all DA numbers and normalized URLs from all pages.

Process:

1. Initializes empty results list and seen_das set for duplicate prevention
2. For each page from 1 to total_pages:
   a. Parses HTML to find results table
   b. Iterates through each row
   c. Extracts DA number from column 2
   d. Checks if DA already seen (skips duplicates)
   e. Extracts href from Show button
   f. Normalizes URL to absolute form
   g. Stores {"da": DA, "url": normalized_url}
   h. Clicks Next button to go to next page
3. Stops when reaching last page

Returns: List of dicts with structure: `[{"da": "PCD25/1535", "url": "https://..."}, ...]`

Duplicate Prevention:

- Maintains `seen_das` set

- Skips any DA number already processed
- Prevents duplicate records in output

Example Data Structure:

```
[
    {"da": "PCD25/1535", "url": "https://...?key=732614"},
    {"da": "PCD25/1536", "url": "https://...?key=732616"},
    {"da": "PCD25/1537", "url": "https://...?key=732618"}
]
```

Example Output:

```
STEP 5: Collecting DA Numbers + Full URLs
 Page 1/22: Collecting... ✓ 10 unique (Total: 10)
 Page 2/22: Collecting... ✓ 10 unique (Total: 20)
...
 Reached last page (22/22)
o Collection Complete: 215 unique DAs
```

D. Data Extraction & Cleaning Functions

`extract_details_from_page(driver)`

Purpose: Extracts all 12 data fields from detail page and applies cleaning rules.

Process:

1. Waits for detail content to load
2. Clicks Expand All button to reveal hidden sections
3. Parses HTML with BeautifulSoup
4. Extracts content from div IDs:
    - `lblDetails` → Description + Submitted Date
    - `lblDecision` → Decision
    - `lblCat` → Categories
    - `lblProp` → Property Address
    - `lblPeople` → Applicant
    - `lblProg` → Progress
    - `lblFees` → Fees
    - `lblDocs` → Documents
    - `lbl91` → Contact Council
5. Parses DA number from page source using regex

6. Parses composite fields:
    - Splits Details by "Submitted:" to get Description and Date
    - Cleans HTML from Properties field
    - Removes "Applicant:" prefix from People field
7. **APPLIES CLEANING RULES**:
    - Fees: If exactly matches FEES_NO_FEES_TEXT, replace with "Not required"
    - Contact_Council: If exactly matches CONTACT_NO_EXHIBITION_TEXT, replace with "Not required"
8. Returns complete record dict

Returns: Dict with 12 fields (all 12 HEADERS) or None if error.

Cleaning Rules Detail:

Rule 1 - Fees Field:

```
if record["Fees"].strip() == "No fees recorded against this application.":
    record["Fees"] = "Not required"
```

- Must be EXACT character-for-character match
- Leading/trailing whitespace stripped before comparison
- Replaces with "Not required" if match

Rule 2 - Contact_Council Field:

```
if record["Contact_Council"].strip() == "Application Is Not on exhibition, please call Co
    record["Contact_Council"] = "Not required"
```

- Must be EXACT character-for-character match
- Replaces with "Not required" if match
- All other contact information preserved as-is

Example Record Output:

```
{
    "DA_Number": "PCD25/1535",
    "Detail_URL": "https://...?key=732614",
    "Description": "New dwelling on flood prone land",
    "Submitted_Date": "01/09/2025",
    "Decision": "Approved 25/09/2025",
    "Categories": "Private Certifier Complying Development Application",
    "Property_Address": "134 Old Southern Rd, WORRIGEE NSW 2540",
    "Applicant": "Bacchus Partners Pty Ltd",
    "Progress": "Completed",
    "Fees": "Not required",  # Cleaned
    "Documents": "PCD25/1535 DA Certificate",
```

```
    "Contact Council": "Not required"  # Cleaned
}
```

`scrape_all_records(driver, da_url_list)`

Purpose: Scrapes all records sequentially from collected DA+URL list with duplicate prevention.

Process:

1. Initializes empty records list and seen_das set
2. For each item in da_url_list:
   a. Extracts DA number and URL
   b. Checks if DA in seen_das (skips if duplicate)
   c. Navigates to URL using `driver.get()`
   d. Calls `extract_details_from_page()` to extract data
   e. If extraction successful:
      - Fills in DA_Number if missing
      - Fills in Detail_URL if missing
      - Appends to records list
      - Marks DA as seen
3. Returns all extracted records

Returns: List of extracted record dicts.

Duplicate Prevention:

- Maintains `seen_das` set throughout process
- Skips any DA already processed
- Logs "duplicate, skipped" for skipped records

Example Output:

```
STEP 6: Scraping All Records Sequentially
[  1/215] DA: PCD25/1535  ✓
[  2/215] DA: PCD25/1536  ✓
[  3/215] DA: PCD25/1537  ✓
...
[215/215] DA: MA25/1322   ✓

o Scraping Complete
   Total records extracted: 215
   Duplicates prevented: 0
```

E. Output & Persistence Functions

`save_records_to_csv(records, filename)`

Purpose: Saves extracted records to CSV file with exact 12 headers in specified order.

Process:

1. Creates pandas DataFrame from records list
2. Ensures all 12 header columns exist (adds empty columns if missing)
3. Reorders DataFrame columns to match exact HEADERS order
4. Saves to CSV using UTF-8-sig encoding (for Excel compatibility)
5. Generates quality report

Returns: True if successful, False if failed.

CSV Specifications:

- **Encoding**: UTF-8 with BOM (utf-8-sig)
- **Headers**: Exact order and spelling as defined in HEADERS list
- **Delimiter**: Comma (,)
- **Index**: Not included

Example CSV Output:

```
DA_Number,Detail_URL,Description,Submitted_Date,Decision,Categories,Property_Address,Appl
PCD25/1535,https://...,New dwelling on flood prone land,01/09/2025,Approved 25/09/2025,Pr
```

Quality Report:

```
Data Quality:
  Non-empty DA_Numbers: 215/215
  Records with Description: 215/215
  Records with Decision: 215/215
  Fees (cleaned): 215/215
  Contact_Council (cleaned): 215/215
```

F. Main Execution Function

`main()`

Purpose: Orchestrates all 7 steps of the scraping pipeline.

Process:

1. Creates WebDriver instance

2. Executes STEP 1: Accept disclaimer
3. Executes STEP 2: Navigate to DA Tracking
4. Executes STEP 3: Open Advanced Search
5. Executes STEP 4: Set date range
6. Executes STEP 4B: Click Search
7. Executes STEP 5: Collect DA + URLs
8. Executes STEP 6: Scrape all records
9. Executes STEP 7: Save to CSV
10. Generates summary report
11. Closes WebDriver

Error Handling:

- Each step returns boolean success/failure
- If any step fails, execution stops gracefully
- Prints detailed error messages
- Always closes WebDriver in finally block

Output Summary:

```
o SCRAPING COMPLETE - SUCCESS
   Expected Records: 215
   Actual Records: 215
   Success Rate: 100.0%
   DA+URLs Collected: 215
   Pages Processed: 22
   Duplicates Prevented: ✓
   Data Cleaning: Applied ✓
   CSV Format: 12 exact headers ✓
```

6. Data Processing Pipeline

Complete Flow Diagram

```
START
  ↓
[STEP 1] Accept Disclaimer
  ↓ (if successful)
[STEP 2] Navigate to DA Tracking
  ↓ (if successful)
[STEP 3] Open Advanced Search
  ↓ (if successful)
[STEP 4] Set Date Range (01/09/2025 - 30/09/2025)
  ↓ (if successful)
[STEP 4B] Execute Search
```

```
  ↓ (if successful)
Extract Total Pages (22) &amp;amp; Items (215)
  ↓ (if extracted)
[STEP 5] Collect DA + URLs from All 22 Pages
  ├─ Page 1: 10 records → Normalize URLs → Add to list
  ├─ Page 2: 10 records → Normalize URLs → Add to list
  ├─ Page 3: 10 records → Normalize URLs → Add to list
  ├─ ...
  └─ Page 22: 5 records → Normalize URLs → Add to list
  ↓ (Result: 215 unique DA+URL pairs)
[STEP 6] Scrape Each Record Sequentially
  ├─ Record 1: Navigate → Extract 12 fields → Apply cleaning → Store
  ├─ Record 2: Navigate → Extract 12 fields → Apply cleaning → Store
  ├─ Record 3: Navigate → Extract 12 fields → Apply cleaning → Store
  ├─ ...
  └─ Record 215: Navigate → Extract 12 fields → Apply cleaning → Store
  ↓ (Result: 215 cleaned records)
[STEP 7] Save to CSV with 12 Headers
  ├─ Create DataFrame
  ├─ Ensure columns: DA_Number, Detail_URL, Description, ...
  ├─ Save to results.csv
  └─ Generate quality report
  ↓ (Result: results.csv with 215 rows + 1 header row)
Generate Summary Report
  ↓
COMPLETE
  ↓
Close WebDriver
  ↓
END
```

7. Data Cleaning Rules (Step 6 Details)

Why Data Cleaning is Critical

Raw extracted data may contain default messages or template text that don't represent actual values. These must be standardized to "Not required" for consistent reporting.

Rule 1: Fees Field Cleaning

Trigger Condition:

- Extracted Fees text equals exactly: `"No fees recorded against this application."`

Action:

- Replace with: `"Not required"`

Logic:

```
if record["Fees"].strip() == "No fees recorded against this application.":
    record["Fees"] = "Not required"
```

Why Strip?: Removes leading/trailing whitespace before comparison, but only whitespace. Character comparison is still exact.

Example:

- Raw: `"No fees recorded against this application."`
- Cleaned: `"Not required"`

Rule 2: Contact Council Field Cleaning

Trigger Condition:

- Extracted Contact_Council text equals exactly: `"Application Is Not on exhibition, please call Council on 1300 293 111 if you require assistance."`

Action:

- Replace with: `"Not required"`

Logic:

```
if record["Contact_Council"].strip() == "Application Is Not on exhibition, please call Co
    record["Contact_Council"] = "Not required"
```

Why This Rule?:

- When application is not on public exhibition, this default message appears
- Needs standardization for reporting purposes
- Shows "Not required" to indicate exhibition is not applicable

Example:

- Raw: `"Application Is Not on exhibition, please call Council on 1300 293 111 if you require assistance."`
- Cleaned: `"Not required"`

All Other Fields

No cleaning applied. All other fields stored exactly as extracted from website:

- Description: Raw text as-is
- Decision: Raw text as-is
- Property_Address: Raw text as-is
- Etc.

8. Output Specification

CSV File Format

File Name: `results.csv`

Encoding: UTF-8 with BOM (ensures Excel compatibility)

Delimiter: Comma (,)

Headers (exactly as specified):

```
DA_Number,Detail_URL,Description,Submitted_Date,Decision,Categories,Property_Address,Appl
```

Row Count: 216 (1 header row + 215 data rows)

Example Data Row:

```
PCD25/1535,https://www3.shoalhaven.nsw.gov.au/masterviewUI/modules/ApplicationMaster/defa
```

Data Integrity Checks

1. **No Duplicates**: All 215 records have unique DA_Number
2. **All URLs Valid**: All URLs are absolute and accessible
3. **All Required Fields**: No empty critical fields (DA_Number, Detail_URL)
4. **Cleaning Applied**:
   - Fees field shows "Not required" for no-fee applications
   - Contact_Council field shows "Not required" for non-exhibition applications
5. **CSV Valid**: File can be opened in Excel, Google Sheets, Python pandas

Excel Opening

To open in Excel:

1. Right-click results.csv
2. Open With → Microsoft Excel
3. Data will auto-import with correct columns and types

9. Error Handling

Try-Except Blocks

The code includes comprehensive error handling at multiple levels:

Level 1: Step Functions

```
if not step_1_accept_disclaimer(driver):
    return  # Stops execution if step fails
```

Level 2: Inner Functions

```
try:
    agree_btn = wait.until(EC.element_to_be_clickable(...))
    agree_btn.click()
except Exception as e:
    print(f"⚠ Agree button not found: {e}")
    return True  # Graceful handling
```

Level 3: Data Extraction

```
try:
    extract_details_from_page(driver)
except Exception as e:
    print(f"✗ ({type(e).__name__})")
    continue  # Skip to next record
```

Common Errors & Recovery

Error	Cause	Recovery
TimeoutException	Page took too long to load	Increases wait time or skips element
NoSuchElementException	Element not found on page	Continues to next step or record
InvalidArgumentException	Invalid URL passed to driver.get()	Skips malformed URL
Stale Element Reference	Element changed after selection	Retries or skips record

WebDriver Cleanup

```
finally:
    driver.quit()  # Always closes browser, even on error
    print("✓ WebDriver closed")
```

Ensures browser process is terminated properly, preventing resource leaks.

10. Usage Guide

Basic Usage

```
# 1. Install dependencies
pip install selenium webdriver-manager beautifulsoup4 pandas

# 2. Run the scraper
python scraper_final_complete_v2.py

# 3. Wait for completion (approximately 30-45 minutes for 215 records)

# 4. Check results
# Open results.csv in Excel or any spreadsheet application
```

Advanced Usage

Running in Headless Mode

```
# Modify main() function:
driver = create_driver(headless=True)  # Runs without showing browser
```

Changing Date Range

```
# Modify configuration at top of file:
START_DATE = "01/08/2025"
END_DATE = "31/08/2025"
```

Changing Output Filename

```
# Modify configuration:
OUTPUT_CSV = "custom_filename.csv"
```

Monitoring Progress

The scraper provides real-time console output:

```
STEP 1: Accept Disclaimer
o Clicked 'Agree' button

STEP 2: Navigate to DA Tracking
o Navigated to DA Tracking

...

STEP 5: Collecting DA Numbers + Full URLs
```

```
 Page 1/22: Collecting... ✓ 10 unique (Total: 10)
 Page 2/22: Collecting... ✓ 10 unique (Total: 20)

[Progress indication for each page]

STEP 6: Scraping All Records Sequentially
[  1/215] DA: PCD25/1535   ✓
[  2/215] DA: PCD25/1536   ✓
[  3/215] DA: PCD25/1537   ✓

[Progress indication for each record]
```

Troubleshooting

Problem: "Chrome not found" error

- **Solution**: Install Google Chrome or run `webdriver-manager` update

Problem: Scraper hangs on specific page

- **Solution**: Increase WAIT_TIME configuration value (e.g., 30 seconds)

Problem: CSV missing some records

- **Solution**: Check console for error messages, rerun scraper

Problem: Links don't navigate correctly

- **Solution**: Verify BASE_APP and BASE_ROOT URLs match current website structure

Summary

This Shoalhaven DA Tracking Web Scraper provides a robust, production-ready solution for automated data collection from the council's DA tracking system. It combines:

- **Reliability**: Comprehensive error handling and recovery
- **Accuracy**: Exact data cleaning rules and duplicate prevention
- **Completeness**: Processes all 215 records from all 22 pages
- **Maintainability**: Well-commented code with clear structure
- **Usability**: Simple configuration and intuitive operation

The scraper successfully demonstrates web automation, data extraction, cleaning, and structured output generation—essential skills for modern data engineering workflows.

Appendix: Code Statistics

- **Total Lines**: ~700
- **Functions**: 13 main functions
- **Comments**: ~150 comment lines

- **Imports**: 11 libraries
- **Configuration Variables**: 12
- **Data Fields Extracted**: 12
- **Processing Steps**: 7
- **Data Cleaning Rules**: 2

Document Version: 1.0
Last Updated: November 1, 2025
Author: Web Scraping Team
