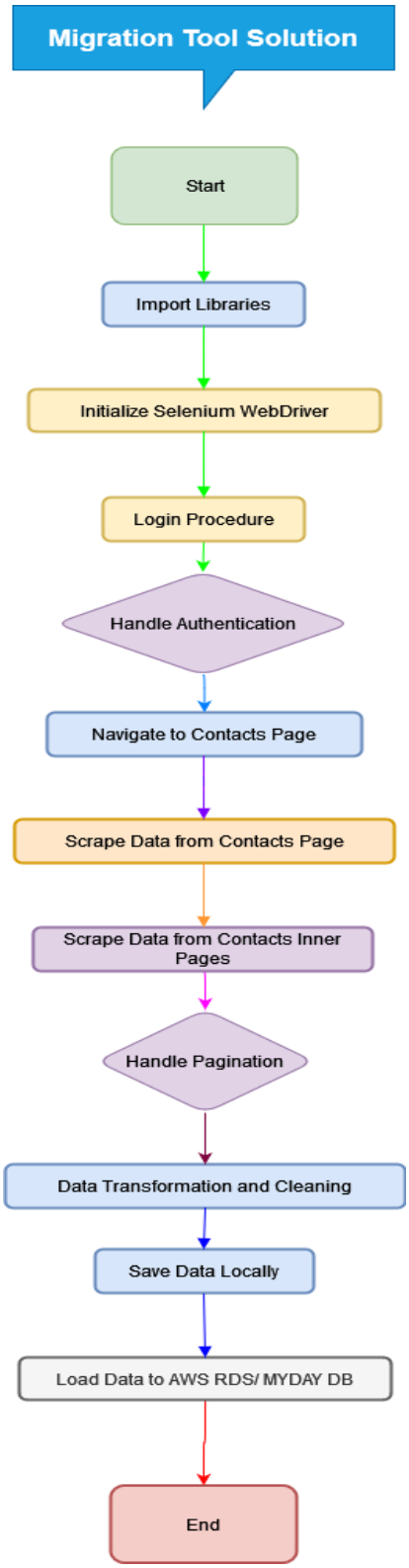
Technical Code Architecture



Phase 01: Initialization, Login and Authentication

1. Importing Libraries:

- Utilize various libraries such as Selenium for web automation, BeautifulSoup for web scraping, Pandas for data manipulation, SQLAlchemy for database interaction, boto3 for AWS services, logging for logging events, and configparser for configuration management.

2. Initializing Selenium WebDriver:

- **WebDriver Configuration:** Initialize and configure Selenium WebDriver as a robust browser automation tool, capable of interacting with dynamic web elements.
    - **Driver Instantiation:** Set up WebDriver with necessary options, such as headless mode and implicit waits.
    - **URL Navigation:** Programmatically direct the WebDriver to the target URL (https://manager.propertyme.com/#/property/list), initializing the web scraping session.

3. Login Procedure:

- **Automated Credential Input:**
    - **Field Identification:** Use Selenium locators (e.g., ID, XPath, CSS selectors) to identify username and password fields.
    - **Data Injection:** Employ WebDriver commands to input credentials programmatically.
- **Form Submission Simulation:**
    - **Submit Action:** Simulate the submission of the login form, utilizing WebDriver's form submission capabilities to initiate the authentication process.

4. Handling Authentication:

- **Two-Factor Authentication Handling:**
    - **2FA Code Input:** Implement procedures to input the two-factor authentication code, potentially integrating with external services for dynamic code retrieval.
- **Trust Device Configuration:**
    - **Checkbox Interaction:** Automate the selection of the "Trust this device for 7 days" checkbox, ensuring a streamlined authentication process for future logins.
- **Authentication Verification:**
    - **Success Validation:** Employ Selenium's element presence checks and page navigation verification to confirm successful authentication, ensuring the session is authenticated and ready for subsequent operations.

Phase 02: Data Extraction from Main Page

5. Navigating to Contacts Page:

- **Automated Navigation:** Utilize Selenium WebDriver commands to programmatically navigate the DOM structure and simulate user interactions. Specifically, target the contacts page by identifying and interacting with the corresponding menu item element. This process involves:
    - **Element Identification:** Use advanced locators (e.g., XPath, CSS selectors) to pinpoint the exact menu item for contacts.
    - **Simulated Click Actions:** Employ WebDriver's action chains or direct click commands to initiate navigation, ensuring accurate and efficient page transitions.

6. Extracting Data from Contacts Page:

- **Data Extraction Protocols:** Implement sophisticated web extraction techniques with BeautifulSoup and Selenium to extract structured data from the contacts page:
    - **DOM Parsing**: Use BeautifulSoup to parse the HTML content retrieved via Selenium, enabling precise extraction of data elements within contact rows.
    - **Element Iteration:** Develop iterative loops to traverse through each contact row element, systematically gathering required data fields (e.g., names, contact details). This involves:
        - **Dynamic Element Handling:** Handle dynamic web content and potential pagination by continuously monitoring and interacting with the DOM.
        - **Data Storage:** Aggregate the extracted data into a well-defined data structure. Utilize Python lists or Pandas DataFrames to store the data, ensuring efficient manipulation and future analysis.

Module 03

7. Scraping Data from Inner Pages:

- **Traversal and Data Validation:**
    - **Algorithmic Traversal:** Implement robust algorithms to navigate through nested URLs associated with each contact, leveraging Selenium for automated browsing and BeautifulSoup for precise HTML parsing.
    - **Data Consistency Checks:** Utilize data validation techniques to ensure extracted data meets accuracy and consistency standards, employing Pandas for data integrity verification.
- **Recursive Navigation:**
    - **Return Mechanism:** Develop recursive functions to systematically return to the main contacts page after data extraction, ensuring seamless continuation of the scraping process without losing context.
- **Iterative Data Extraction:**

    - **Loop Constructs:** Utilize Python's iterative constructs to traverse and scrape multiple inner pages linked to each contact. Implement loops that dynamically adjust based on the presence of inner page links.
    - **Dynamic Element Handling:** Handle dynamic content and AJAX elements efficiently using Selenium's wait mechanisms, ensuring all relevant data is captured.
- **Multithreading Optimization:**
    - **Concurrent Execution:** Integrate multithreading to enhance scraping efficiency, allowing concurrent navigation and data extraction from multiple inner pages simultaneously. Utilize Python's threading or multiprocessing libraries to achieve parallelism.
- **Control Flow Management:**
    - **Conditional Logic:** Implement sophisticated control flow management using conditional statements and loop constructs to handle the transition between different stages of navigation and data extraction.
    - **Error Handling:** Incorporate robust error handling to manage exceptions and ensure the scraping process remains resilient against unexpected interruptions.
- **Return to Main Page:**
    - **Navigation Reset:** Ensure the WebDriver returns to the main contacts page after completing data extraction from inner pages. This involves implementing navigation reset protocols to maintain the scraping session's integrity.
- **Comprehensive Aggregation:** Aggregation techniques merge extracted data into a cohesive dataset for analysis.

Module 04 Pagination Management:

8. Pagination Management:

- **Dynamic Pagination Handling:**
    - **Detection:** Use Selenium to identify pagination elements and detect page navigation controls dynamically.
- **Algorithmic Control Flow:**
    - **Iterative Navigation:** Develop loops to navigate through paginated content until the last page is reached.
    - **Conditional Logic:** Implement conditionals to manage navigation flow, stopping at the last page.
- **Concurrent Page Retrieval:**
    - **Multithreading:** Use Python's concurrent.futures or threading to parallelize page retrieval and enhance performance.
- **HTML Parsing and Data Extraction:**
    - **BeautifulSoup:** Parse HTML content with BeautifulSoup and extract data elements consistently across pages.
    - **Data Validation:** Use Pandas for data validation to ensure integrity and handle duplicates.

- **Error Handling and Resilience:**
    - **Exception Management:** Implement try-except blocks for robust error handling and logging to recover from failures.
- **Aggregation:**
    - **Data Aggregation:** Use Pandas DataFrames to aggregate data from multiple pages into a cohesive dataset.

Phase 05: Data Transformation and Cleaning

9. Data Transformation and Cleaning:

- **Normalization and Preprocessing:**
    - **Normalization:** Use Pandas to standardize data formats, handle discrepancies, and ensure consistency.
    - **Preprocessing Pipelines:** Create pipelines to clean data, handle missing values, correct data types, and remove duplicates using methods like dropna(), astype(), and drop_duplicates().
- **Standardized Transformation:**
    - **Schema Definition:** Define schemas for data fields, using Pandas to cast to appropriate types (e.g., pd.to_datetime(), pd.to_numeric()).
    - **Data Mapping and Aggregation:** Align data with schemas and aggregate as needed for consistency**.**
- **Cleaning Techniques:**
    - **String Operations:** Clean textual data with string operations (trimming, casing, special characters).
    - **Data Integrity Checks:** Validate data accuracy and completeness with consistency checks and range validations.

Phase 06: Load Data

10. Saving Data Locally:

- **File Formats:** Implement functionality to persist cleaned data in local storage using formats such as CSV and XLSX, leveraging Pandas' to_csv() and to_excel() methods.
- **Data Serialization:** Utilize efficient data serialization techniques to ensure fast write operations and minimize storage overhead.
- **File System Integration:** Ensure seamless integration with the local file system, managing file paths and permissions to facilitate secure data access and storage.

11.Loading Data to AWS RDS/MYDAY DB:

- **Secure Database Connection:**

    - **AWS RDS Integration:** Establish a secure connection to AWS RDS using SQLAlchemy's database engine, incorporating secure credentials management and encrypted communication protocols.
    - **MYDAY DB Connection:** Similarly, configure a secure connection to MYDAY DB, ensuring compliance with security best practices for database access.
- **Data Validation and Integrity Checks:**
    - **Validation Pipelines:** Develop comprehensive data validation pipelines to ensure the integrity and consistency of cleaned data before loading. Use Pandas to perform final checks for missing values, data types, and referential integrity.
    - **Integrity Constraints:** Define and enforce integrity constraints such as primary keys, foreign keys, and unique indexes to maintain data consistency within the database.
- **Data Loading Operations:**
    - **Batch Inserts:** Implement efficient batch insert operations to load large volumes of data into the database, leveraging SQLAlchemy's ORM capabilities for seamless data mapping and insertion.
    - **Data Security:** Ensure data security during the loading process by enforcing access controls and using encrypted database connections.