Technical Documentation

Overview

This project uses several libraries and modules for web automation, file handling, and time manipulation, I used Python 3.12.5 version. Below are the key libraries and their purposes:

Libraries:

time: Provides various time-related functions.

```
import time
```

pyautogui: Allows automation of mouse and keyboard control.

```
import pyautogui
```

selenium.webdriver.support.expected_conditions as EC: Defines expected conditions for WebDriver operations.

```
from selenium.webdriver.support import expected_conditions as EC
```

selenium.webdriver.support.ui.WebDriverWait: Waits for a condition to be met within a specified period.

```
from selenium.webdriver.support.ui import WebDriverWait
```

selenium.webdriver.common.by.By: Methods to locate elements on the page.

```
from selenium.webdriver.common.by import By
```

selenium.common.exceptions.NoSuchElementException: Exception for missing elements.

```
from selenium.common.exceptions import NoSuchElementException
```

webdriver_manager.chrome import ChromeDriverManager: Manages download and setup of ChromeDriver.

```
from webdriver_manager.chrome import ChromeDriverManager
```

selenium.webdriver.chrome.options.Options: Sets options for ChromeDriver.

```
from selenium.webdriver.chrome.options import Options
```

selenium.webdriver.chrome.service.Service: Starts and stops ChromeDriver as a service.

```
from selenium.webdriver.chrome.service import Service
```

webdriver: Main module for Selenium WebDriver.

```
from selenium import webdriver
```

openpyxl: Manipulates Excel files.

```
import openpyxl
```

os: Interfaces with OS-dependent functionalities.

```
import os
```

datetime: Handles dates and times.

```
from datetime import datetime
```

Variables

url_site: Placeholder for the target website URL.

```
url_site = 'Insert your URL site'
```

url_login: Placeholder for the user login URL.

```
url_login = 'Insert your user'
```

your_user: Placeholder for the username.

```
your_user = "Insert your user"
```

access_key: Placeholder for the password.

```
access_key = "Insert your password"
```

invoices_with_error: An empty list to store invoices that encounter errors.

```
invoces_with_error = []
```

Functions

Functions load_dataframe

Description: Loads data from an Excel sheet into a list of dictionaries.

Code:

```python
def load_dataframe(file_name="sheet.xlsx", sheet_name="Insert_your_sheet_page"):
    current_directory = os.path.dirname(os.path.abspath(__file__))
    file_path = os.path.join(current_directory, file_name)
    try:
        workbook = openpyxl.load_workbook(file_path)
        sheet = workbook[sheet_name]
        data = []
        headers = [cell.value for cell in sheet[1]]
        for row in sheet.iter_rows(min_row=2, values_only=True):data.append(dict(zip(headers, row)))
        num_items = len(data)
        print(f"Base with {num_items} CNPJ's")
        return data
    except FileNotFoundError:
        print(f"Error: File '{file_name}'not found.")
        return None
```

Parameters:

file_name (str): Name of the Excel file. Default is "sheet.xlsx".

sheet_name (str): Name of the sheet in the Excel file. Default is "Insert_your_sheet_page".

Returns: List of dictionaries containing the data from the Excel sheet, or None if the file is not found.

Example:

```python
data = load_dataframe(file_name="data.xlsx", sheet_name="Sheet1")
```

Detailed Explanation:

The function gets the current directory path and constructs the full file path.
It tries to load the workbook and the specified sheet.
It reads the headers from the first row and the data from subsequent rows.
It prints the number of items (rows) loaded.
If the file is not found, it prints an error message and returns None.

Function setup_driver

Description: Configures and initializes a Chrome WebDriver instance with specific options.

code:

```
def setup_driver():

    chrome_options = webdriver.ChromeOptions()
    chrome_options = Options()
    chrome_options.add_argument('--remote-debugging-port=9666')
    chrome_options.add_argument('--ignore-certificate-errors')
    driver = webdriver.Chrome( options=chrome_options)
    driver.set_window_size(1366, 768)
    return driver
```

Returns:

driver: A Chrome WebDriver instance with the configured options.

Example:

```
driver = setup_driver()
```

Detailed Explanation:

- Creates an instance of Options for ChromeDriver.
- Adds the option for remote debugging on port 9666 and to ignore certificate errors.
- Initializes the Chrome WebDriver with the specified options.
- Sets the window size to 1366x768 pixels.

Returns the configured driver instance.

Function login

Description: Automates the login process on a website and navigates to a specified URL.

Code:

```python
def login(driver, url_login, your_user, access_key, url_site):
    try:
        driver.get(url_login)
        driver.find_element(By.XPATH, 'insert here your xpath referring to the user field').send_keys(your_user)
        driver.find_element(By.XPATH, 'insert here your xpath referring to the key field').send_keys(access_key)
        driver.find_element(By.XPATH, 'insert here your xpath referring to the button to enter').click()
        driver.get(url_site)
        iframe = driver.find_element(By.XPATH, 'insert your iframe or xpath of your search box element here')
        driver.switch_to.frame(iframe)
    except Exception as e:
        print(f"Error when you logging: {e}")
```

Parameters:

- driver: A Chrome WebDriver instance.
- url_login (str): URL for the login page.
- your_user (str): Username for login.
- access_key (str): Password for login.
- url_site (str): Target URL to navigate to after login.

Example:

```python
login(driver, url_login, your_user, access_key, url_site)
```

Detailed Explanation:

Navigates to the login URL.

Locates and inputs the username and password in their respective fields using XPath.

Clicks the login button.

Navigates to the target URL.

Locates an iframe or search box element and switches the driver's context to that iframe.

Handles exceptions by printing an error message if the login process fails.

Function search_cnpj

Description: Searches for a CNPJ (Cadastro Nacional da Pessoa Jurídica) using the web interface and interacts with the search results.

Code:

```
def search_cnpj(driver, cnpj):
    wait = WebDriverWait(driver, 30)
    try:
        element = wait.until(EC.element_to_be_clickable( (By.XPATH, 'insert here your xpath referring to the button top search')))
        element.clear()
        element.send_keys(cnpj)
        element = wait.until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath reffering result')))
        elemento.click()
        elemento = wait.until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath')))
        elemento.click()
```

Parameters:

driver: A Chrome WebDriver instance.

cnpj (str): The CNPJ to search for.

Example:

```
search_cnpj(driver, "your_cnpj_here")
```

search_cnpj:
Description: Searches for a CNPJ (Cadastro Nacional da Pessoa Jurídica) using the web interface and interacts with the search results.

Parameters:

driver: A Chrome WebDriver instance.

cnpj (str): The CNPJ to search for.

Example:

- search_cnpj(driver, "your_cnpj_here")

Detailed Explanation:

- Waits for up to 30 seconds for the search button element to be clickable.
- Clears the search field and enters the CNPJ value.
- Waits for the search result element to be clickable.
- Clicks the search result element.
- Waits for another specified element to be clickable and clicks it.

Error Handling:
Handles exceptions by printing an error message if any step of the search process fails.

Function click_verify_download_invoice

Description: This function searches for and verifies invoices on a web page, then downloads them if found.

Code:

```python
def clicK_verify_download_invoice(driver, base_invoce_number,document):
    wait = WebDriverWait(driver, 30)

    table_rows= driver.find_elements(By.XPATH, f"insert here your xpath")
    quantity_of_items = len(table_rows)
    print(f"Number of records found: {quantity_of_items}")

    for i in range(2, quantity_of_items):

        xpath = f'insert here your xpath {i} generator'

        try:

            wait.until(EC.element_to_be_clickable((By.XPATH, xpath))).click()
            print(f"Button found {i}º registration")
            def get_number_note():
                invoce = WebDriverWait(driver, 20).until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath')))
                numero_nota_web = invoce.text
                return numero_nota_web
```

Parameters:

- driver: A Chrome WebDriver instance.
- base_invoice_number (str): The base number of the invoice to verify.
- document (str): Additional document data, if necessary.

Example:

```
click_verify_download_invoice(driver, base_invoice_number, document)
```

Detailed Explanation:

Initialize WebDriverWait:

- wait = WebDriverWait(driver, 30): Sets up a wait condition to handle loading times, waiting up to 30 seconds for elements to be clickable.

Find Table Rows:

- table_rows = driver.find_elements(By.XPATH, f"insert here your xpath"): Locates the table rows on the webpage using the specified XPath.

- quantity_of_items = len(table_rows): Counts the number of table rows found.
- print(f"Number of records found: {quantity_of_items}"): Prints the number of records found.

Iterate Through Table Rows:

- for i in range(2, quantity_of_items): Loops through each table row starting from the second row to avoid headers.
- xpath = f'insert here your xpath {i} generator': Generates the XPath for each row.

Click Elements:

- try: wait.until(EC.element_to_be_clickable((By.XPATH, xpath))).click(): Waits for the element to be clickable and then clicks it.
- print(f"Button found {i}º registration"): Prints a message indicating which button was clicked.

Sub-function get_number_note:

- def get_number_note(): Defines an inner function to get the invoice number.
- invoce = WebDriverWait(driver, 20).until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath'))): Waits up to 20 seconds for the invoice element to be clickable.
- numero_nota_web = invoce.text: Extracts the text of the invoice number.
- return numero_nota_web: Returns the invoice number.

Error Handling:

The try block ensures that if an element is not clickable or another error occurs, it will continue to attempt to find and click elements without stopping the entire process.

Function formation_data

Description: This function formats a given date string into the DDMMYYYY format.

Code:

```
def formation_data(emission_data):
            formation = ['%Y-%m-%d %H:%M:%S']

            for formation in formation:
                try:
                    data = datetime.strptime(emission_data, formation)
                    return data.strftime("%d%m%Y")

                except ValueError:

                    raise ValueError(
                        f"Unrecognized date/time format: {emission_data}")
```

Parameters:

emission_data (str): A string representing the date and time to be formatted.

Example:

```
formatted_date = formation_data("2023-09-15 08:30:00")
```

Detailed Explanation:

Define Accepted Format:

- formation = ['%Y-%m-%d %H:%M:%S']: A list containing the expected input date format.

Iterate and Attempt to Parse Date:

- for formation in formation:: Iterates over the list of accepted formats (in this case, just one format).

Try to Parse and Reformat:

- data = datetime.strptime(emission_data, formation): Tries to parse the input string emission_data using the given format.

- return data.strftime("%d%m%Y"): If successful, converts the date to the DDMMYYYY format and returns it.

Handle Invalid Formats:

- except ValueError:: Catches the exception if the date string does not match the expected format.

- raise ValueError(f"Unrecognized date/time format: {emission_data}"): Raises a ValueError with a message indicating the unrecognized date format.

click_verify_download_invoice

Code:

```
try:

    base_invoce_number = str(base_invoce_number).strip()
    document = str(document)
    invoce_number_web = get_number_note()
    formation_data()


    if int(invoce_number_web) == int(base_invoce_number):
        print(str(f"Invoce {base_invoce_number} successfully found."))

        time.sleep(3)
        pyautogui.press('end')
        time.sleep(3)
        pyautogui.press('pagedown')
        time.sleep(3)
        pyautogui.click(704, 642)
        time.sleep(15)
        pyautogui.click(x=1238, y=183)
        time.sleep(5)
        pyautogui.press('enter')
        time.sleep(5)
        pyautogui.hotkey('ctrl', 'w')
        time.sleep(5)

        download_folder = r"insert here your path to repository"
        filename_invoice = os.path.join(download_folder, "download.pdf")

        new_filename_invoice = os.path.join(download_folder, f"DOC_{document}_NF_{base_invoce_number}.pdf")
        os.rename(filename_invoice, new_filename_invoice)
        print(f"File renamed to: {new_filename_invoice}")
        return True
```

Detailed Execution Steps:

Initialize and Clean Input Data:

- base_invoice_number = str(base_invoice_number).strip(): Converts base_invoice_number to a string and removes any leading or trailing whitespace.
-
- document = str(document): Converts document to a string.
-

Retrieve Invoice Number from Web:

- invoice_number_web = get_number_note(): Calls the get_number_note function to retrieve the invoice number from the web page.

Format Date:

- formation_data(): Calls the formation_data function to format the date (not clear from this snippet how formation_data is used).

Compare Invoice Numbers:

if int(invoice_number_web) == int(base_invoice_number): Compares the invoice number retrieved from the web with the base invoice number.

Actions upon Successful Match:

- print(str(f"Invoce {base_invoice_number} successfully found.")): Prints a success message.
- time.sleep(3): Pauses for 3 seconds.
- pyautogui.press('end'): Simulates pressing the "End" key to scroll to the bottom of the page.
- time.sleep(3): Pauses for 3 seconds.
- pyautogui.press('pagedown'): Simulates pressing the "Page Down" key to scroll down the page.
- time.sleep(3): Pauses for 3 seconds.
- pyautogui.click(704, 642): Clicks at coordinates (704, 642) on the screen.
- time.sleep(15): Pauses for 15 seconds to allow the page or action to complete.
- pyautogui.click(x=1238, y=183): Clicks at coordinates (1238, 183) on the screen.
- time.sleep(5): Pauses for 5 seconds.
- pyautogui.press('enter'): Simulates pressing the "Enter" key.
- time.sleep(5): Pauses for 5 seconds.
- pyautogui.hotkey('ctrl', 'w'): Simulates pressing the "Ctrl + W" hotkey to close the current tab.
- time.sleep(5): Pauses for 5 seconds.

Rename and Move the Downloaded File:

- download_folder = r"insert here your path to repository": Specifies the download folder path.
- filename_invoice = os.path.join(download_folder, "download.pdf"): Defines the original filename of the downloaded PDF.

- new_filename_invoice = os.path.join(download_folder, f"DOC_{document}_NF_{base_invoice_number}.pdf"): Generates a new filename using the document type and invoice number.

- os.rename(filename_invoice, new_filename_invoice): Renames the downloaded PDF file to the new filename.
-
- print(f"File renamed to: {new_filename_invoice}"): Prints a message indicating the file has been renamed.

return True: Returns True to indicate successful execution.

Handling Invoice Mismatch and Errors

Description: This part of the code deals with scenarios where the invoice numbers do not match and handles various exceptions.

Detailed Explanation:

Handling Mismatched Invoices:

Conditional Block:

```
else:

    print(str(f"Invoce number: {invoce_number_web}, does not match the desired grade: {base_invoce_number}"))
    element = wait.until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath')))
    element.click()
```

If the invoice numbers do not match, prints a message indicating the mismatch.
Waits until a specified element is clickable and then clicks it.

Handling Missing Invoices (NoSuchElementException):

Exception Block:

```
except NoSuchElementException:
    print(f"Invoce unavailable: {base_invoce_number}")
    invoces_with_error.append([base_invoce_number])
    element = wait.until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath')))
    element.click()
```

- Catches NoSuchElementException if the element is not found.
- Prints a message indicating the invoice is unavailable.
- Appends the base invoice number to the invoices_with_error list.
- Waits until a specified element is clickable and then clicks it.

Handling General Exceptions:

Exception Block:

```
except Exception as e:
    print(f"Error finding the button {i}º registration: {e}")

    element = wait.until(EC.element_to_be_clickable((By.XPATH, 'insert here your xpath')))
    element.click()
```

Catches any other exceptions that may occur.
Prints a message indicating an error in finding the button for the specific registration.
Waits until a specified element is clickable and then clicks it.

error_invoces

```
def error_invoces():
    workbook = Workbook()
    sheet = workbook.active
    sheet.title = "ERROR_INVOCES_UNAVAILABLE"
    for invoce in invoces_with_error:
        sheet.append([invoce])

    workbook.save("ERROR_INVOCES_UNAVAILABLE.xlsx")
    print("Error invoces saved successfully")
```

Description: This function creates an Excel workbook to record invoices that encountered errors during the processing and saves the workbook to a file.

Parameters: None

Example:

```
error_invoces()
```

Detailed Explanation:

Create a New Workbook:

- workbook = Workbook(): Initializes a new workbook using the openpyxl library.
- sheet = workbook.active: Gets the active sheet in the workbook.
- sheet.title = "ERROR_INVOCES_UNAVAILABLE": Sets the title of the active sheet to "ERROR_INVOCES_UNAVAILABLE".

Append Invoices with Errors:

- for invoce in invoces_with_error: sheet.append([invoce]): Iterates over the invoices_with_error list and appends each invoice to the sheet.

Save the Workbook:

- workbook.save("ERROR_INVOCES_UNAVAILABLE.xlsx"): Saves the workbook to a file named "ERROR_INVOCES_UNAVAILABLE.xlsx".

- print("Error invoces saved successfully"): Prints a confirmation message indicating that the invoices with errors have been successfully saved.

This function ensures that all invoices that encountered errors during the processing are logged and saved in an Excel file for further review.

main

Code:

```
def main():
    while True:
        try:
            data = load_dataframe()
            if data is None:
                print("Error: Data is empty.")
                return
            driver = setup_driver()
            login(driver, url_login, your_user, access_key, url_site)
            for row in data:
                cnpj = row['CNPJ']
                base_invoice_number= row['INVOICE']
                document = row ['DOCUMENT']
                emission_data = row['EMISSION']
                search_cnpj(driver, cnpj)
                clicK_verify_download_invoice(
                driver, base_invoice_number,document)
        except Exception as e:
            print(f"Error executing function main: {e}")

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

Description: The main function that coordinates the entire automation process, including loading data, setting up the driver, logging in, and processing each record.

Parameters: None

Example:

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

Detailed Explanation:

Infinite Loop for Continuous Execution:
while True:: Ensures the automation process runs continuously.

Exception Handling for the Entire Process:
try:: Wraps the main logic in a try block to catch and handle any exceptions.

Load Data from Excel:

- data = load_dataframe(): Calls the load_dataframe function to load data from an Excel file.
- if data is None: print("Error: Data is empty.") return: Checks if the data is empty and prints an error message, then exits the function if data is not loaded.

Set Up WebDriver:
- driver = setup_driver(): Calls the setup_driver function to configure and initialize the Chrome WebDriver.

Log In to the Website:

- login(driver, url_login, your_user, access_key, url_site): Automates the login process by calling the login function.

Process Each Row of Data:
- for row in data:: Iterates through each row in the loaded data.

Extracts the relevant fields from each row:

- cnpj = row['CNPJ']: Extracts the CNPJ.
- base_invoice_number = row['INVOICE']: Extracts the base invoice number.
- document = row['DOCUMENT']: Extracts the document type.
- emission_data = row['EMISSION']: Extracts the emission data.

Calls the processing functions for each record:

- search_cnpj(driver, cnpj): Searches for the CNPJ on the website.
- click_verify_download_invoice(driver, base_invoice_number, document): Clicks, verifies, and downloads the invoice.

Exception Handling:
- except Exception as e: print(f"Error executing function main: {e}"): Catches any exceptions that occur during the execution of the main function and prints an error message.

Entry Point for Script Execution:
- if __name__ == '__main__': main(): Ensures that the main function is called when the script is executed directly.

This function orchestrates the entire automation workflow, from loading data and setting up the driver to processing each record and handling exceptions.