

Content

1. Project title
2. Software requirements
3. Foreword
4. Project description
5. Flow diagram
6. How to install, pre-setup and run the Python script
7. How to adapt and use the whole project
8. Python functions and features

1. Project title

ERP data retrieve tool from JD Edwards

2. Software requirements:

- Python 3.8 with Selenium framework installed
- JD Edwards Enterprise One 9.0 (web application)
- Windows / macOS
- Chrome 113.0 or higher

3. Foreword

The application shows possibilities of automatization of ERP data gathering that can be used for multiple purposes within businesses. Idea here is to automate daily tasks which are essentially time-consuming. I have started this mindset after reading the book “Automate the Boring Stuff with Python” written by Al Sweigart and then asking our engineering team what are daily routine tasks which basically they do not like the most. In this example, the script is used for Return Material Authorisation process (RMA) which is a massive workload, especially in supply chain business. Nevertheless, looking at my whole engineering experience with ERP systems I believe it can be easily adapted in multiple business sectors where there is a need to perform thousands of manual ERP transactions every day and ERP is available via web application.

4. Project Description

In this particular project, program allows to find specific information in ERP such as:

- export control data for specific product saved in ERP system under unique Part Number (PN) which is an input data
- unit price for combination of unique Part Number (PN) and Purchase Order (PO) being an input data

In the given program example the script finds and copies an input data from the Excel file, then navigates through the web version of the ERP system, pastes stored input information in the exact fields in ERP (Part Number or both Part Number and Purchase Order), subsequently retrieves multiple data outputs and pastes these data back into another Excel output file (export control data and unit price).

The script is written in Python 3.8 and uses Selenium framework, which allows to navigate through the ERP system. ERP version navigated by this script is JD Edwards Enterprise One 9.0. The JDE system is run as a web version, so the webdriver works within Chrome browser and the webdriver can be easily updated for every new version of Chrome.

Below you can find some of the challenges and features I discovered during the project design:

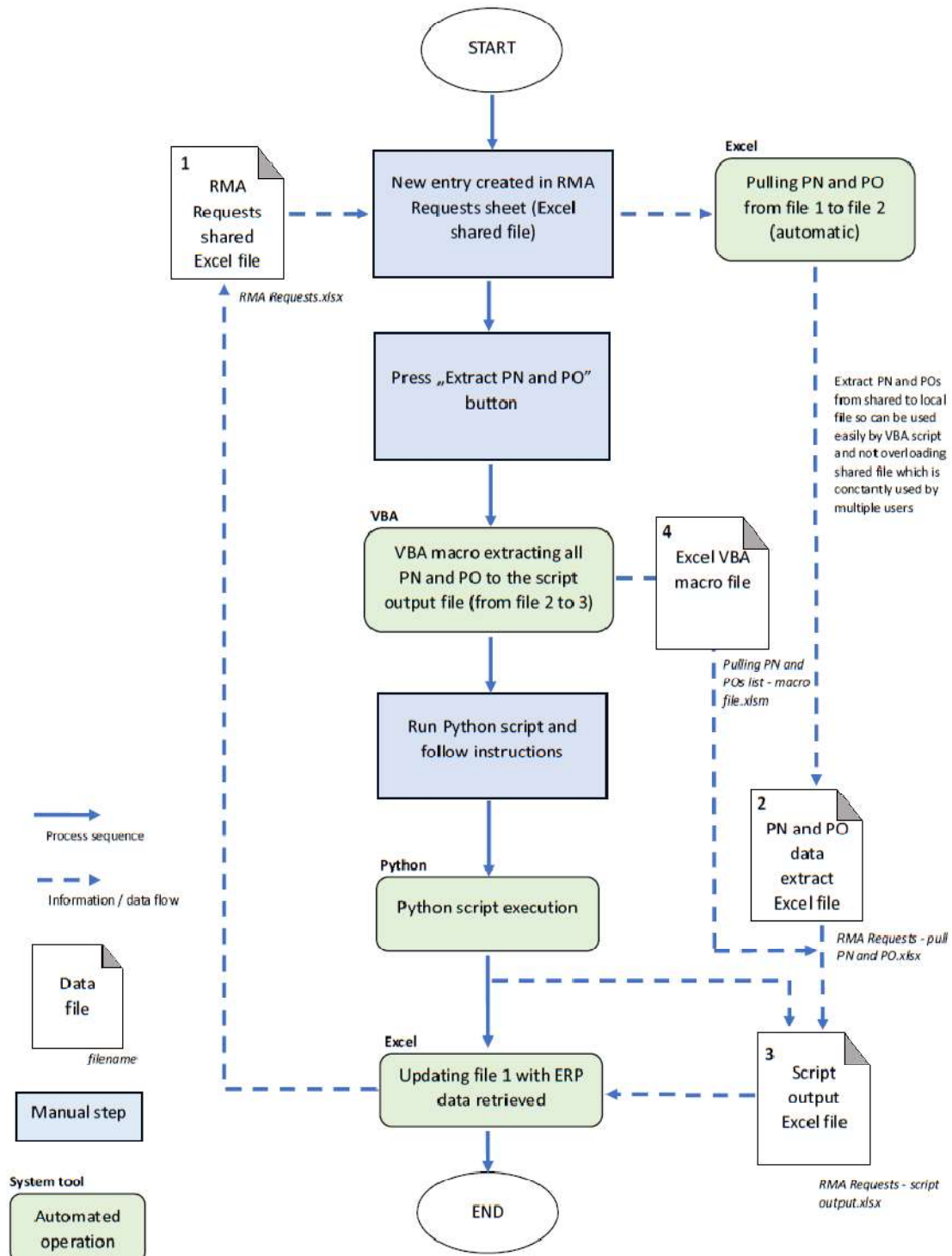
- integration of Python script with two different Excel files (being input and output for the data) – I was not able to handle Excel files with Python, therefore I used simple VBA script to move the data from one file to another
- use an Excel macro file with VBA code as an interface between Excel files
- usage of *dotenv* module and environment variable to store data as a login that is used in ERP application
- password input feature as a separate window that can store and use password used in ERP application

For the successful repository usage and adapting it to create an own automation tool there is a need to install Python with all the packages or libraries and then make simple adaptations based on further instructions described in chapters 6 and 7.

5. Flow diagram

For the explanation of the concept behind the project I created following flow diagram.

Apart from Python files, important part of the system are Excel files which act as input and output for the Python script.

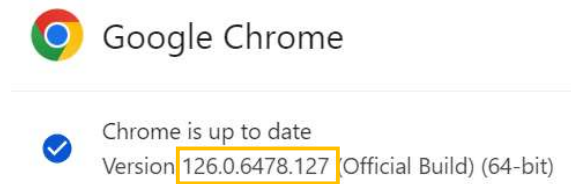


6. How to install, pre-setup and run the Python script

a. Install *ChromeDriver*:

Check your version of Chrome by going into in the browser to this address:

chrome://settings/help

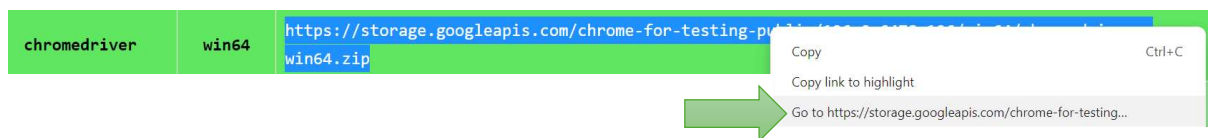


Go to: <https://googlechromelabs.github.io/chrome-for-testing/>

Find matching version of ChromeDriver (126.0 in this case):

Channel	Version	Revision	Status
Stable	126.0.6478.126	r1300313	✓

Click and download version for your system (Windows 64 bit in this case):



Open the package file and copy the application “chromedriver” file



Finally, paste the chromedriver file into: ***C:/Users/Public/***

b. Setup JD Edwards login credentials file

Open Notepad and write in your JD Edwards login in following format. Password field is not needed here as it will be provided separately in another window prompt. For now I will use *kamil_pi* user as a example here.

```
File Edit Format View Help
user = kamil_pi
pass =
```

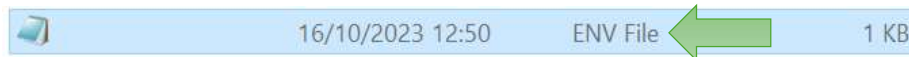
Choose File -> **Save As**

Choose your folder in C:\Users\<**your_personal_folder**>

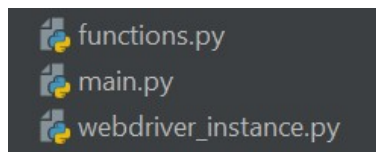


Put name of the file “**.env**”

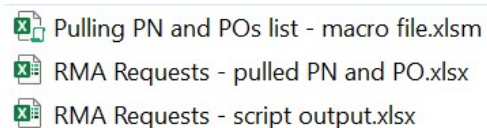
Choose **Save as type: All Files** and save it
Make sure that the file has .ENV extension in the explorer:



- c. Install Python (at least version 3.8)
- d. Within the IDE create a new project and copy the three essential Python .py files from the portfolio:



- e. Copy three Excel files from the repository into the same project or pre-defined folder



- f. Now almost ready to run the project from the IDE shell / terminal by running the **main.py** file. However, before first running of the program, please follow the steps of the chapter 7 to modify essential data inside some of the files and prepare a whole environment. There are a few places that need to be updated before use.

7. How to adapt and use the project

Python and Excel files need some further modification in order the program will run smoothly on your side. Here are the steps:

- a. Go to the *function.py* file and update the user reference number and path to your personal folder under the *user_names* and *env_files* variables. It should look like the example below. I have put myself "kamil_pi" under the user number 3. During script execution, one of the function will match the username number input with dotenv file paths in the user defined folder. This will allow to use stored ERP username shortly afterwards.

```
env_files = {  
    '1': 'C:/Users/user1_name/.env', '2': 'C:/Users/user2_name/.env', '3': 'C:/Users/kamil_pi/.env'  
}  
  
user_names = {  
    '1': 'user1', '2': 'user2', '3': 'kamil_pi'  
}
```

- b. URL to the ERP system: URL for JD Edwards should be updated with the correct path

```
# URL to JD Edwards
url_jd_edwards = "url_to_JD_Edwards"
```

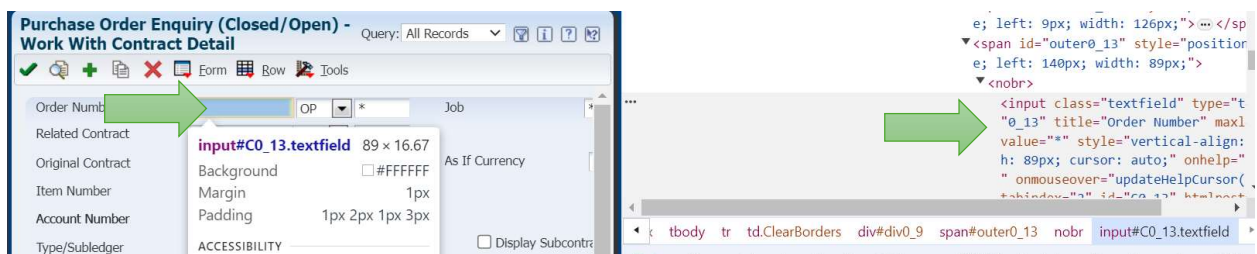
- c. XPATH references in Python *functions.py* – Selenium functions can navigate to the web objects using XPATH references. It is very likely that in other instances of the JD Edwards software XPATHs references may vary and would need to be updated in order to the program to work. However, it is not too problematic as finding XPATH and replacing is relatively easy and takes a few of seconds.

To find the correct XPATH you need to follow these steps:

While having JDE web application window active open the developer tools, click on the inspection icon.



Click left mouse button on the element we want to navigate to. Element reference will appear on the developer tab on the right side.



Click right mouse button on the element reference in the developer tab and choose

Copy > Copy full XPath

Now XPath is copied and you can replace it under desired object in the *python.py* file




```
xpath_cust_pn_empty = "/html/body/form[3]/div/table/tbody/tr/td/div/span[10]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr[2]/td/div/table/tbody/tr/td[2]/table/tbody/tr[6]/td/table/tbody/tr/td/div/input"
```

- d. Python *function.py* file: Excel file reference to the output file update – make sure Excel output file exists and is stored under correct location

```
# URL to the RMA requests output Excel file
url_output_file = "url_to_the_output_file"
```

e. Excel files – Python script from this portfolio works only with the Excel files. You need to configure the files before whole system usage.

f. Ensure all files exist in one place:

-  RMA Requests - script output.xlsx
-  RMA Requests - pulled PN and PO.xlsx
-  PN and PO for output file - macro.xlsm


i. *RMA Requests – pulled PN and PO* – the file (file 2 on the flow diagram) can be a master file with essential Part Number and Purchase Order data pulled from the root file (file 1 on the flow diagram). In other words, the base file 1 from the diagram is optional and we can consider using only file 2 as a base. In my case, the original file 1 was a shared file with multiple users working on it and it was really hard to manipulate by Python. Therefore I “pulled” all the essential data (Part Number and Purchase Order) from file 1 to file 2.

	A	B	C
1	No.	Part Number	Purchase Order
2	1		
3	2		
4	3		

ii. *RMA Requests - script output.xlsx* – all ERP data will land in this file in respective tabs (Raw_JDE and RawJDE2)

The data will present the data in following format.

No. P/N	PO	P/N_2	1st Branch HSC Code:	1st Branch HSC Desc:	Master HSC Code or 2nd branch HSC Code: Part description	Export classification	Unit Cost
Part Number and Purchase Order data transferred by VBA code in the same sequence as in original base file		Part Number modified to ensure all values are TEXT type	Export control data pulled from Raw_JDE tab				



Unit cost data pulled from Raw_JDE2 tab

iii. *PN and PO for output file.xlsm* (macro VBA code) - apart from two standard .xlsx files (2. and 3.) this macro file transfers the data from file 2: *RMA Requests - pulled PN and PO.xlsx* to the file 3: *RMA Requests - script output.xlsx*. Below you may see the snapshot from the VBA code. Please make sure that references and filenames

shown below are correct. You can open the code editor and view the code by opening this macro file, using shortcut Alt + F11 and click on “ThisWorkbook” tab.

```
' Set the file paths
Dim sourceFilePath As String
Dim targetFilePath As String
sourceFilePath = "path_to_the_file: RMA Requests - pulled PN and PO.xlsx"
targetFilePath = "path_to_the_file: RMA Requests - script output.xlsx"

' Open the source workbook from SharePoint
Set sourceWorkbook = Workbooks.Open(sourceFilePath)
If Err.Number <> 0 Then
    MsgBox "Error opening source workbook: " & Err.Description
    Exit Sub
End If

' Set the source sheet
Set sourceSheet = sourceWorkbook.Sheets("Copy_RMA_Requests")

' Open the target workbook
Set targetWorkbook = Workbooks.Open(targetFilePath)
If Err.Number <> 0 Then
    MsgBox "Error opening target workbook: " & Err.Description
    Exit Sub
End If

' Set the target sheet
Set targetSheet = targetWorkbook.Sheets("Values_RMA_Requests")

' Determine the last row with data in the source sheet
lastRow = sourceSheet.Cells(sourceSheet.Rows.Count, "A").End(xlUp).Row
lastColumn = 3 ' Column C

' Copy specific columns (A, B, C) from source to target
sourceSheet.Range("A:C").Copy
targetSheet.Cells.PasteSpecial xlPasteValues
```

- g. Run Python script in IDE shell / terminal by running the “main.py” file and choose the correct user number*

```
---
-----
ERP data retrieve tool from JD Edwards
-----
---

>>>>>

Please specify user to login into JDE or press 0 for EXIT:

1 : User1
2 : User2
3 : kamil_pi
4 : User4
5 : User5
6 : User6
7 : User7
8 : User8

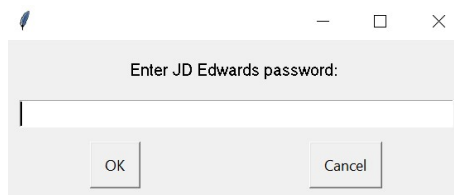
Your choice: |
```


* Note that at the time the program is run, there is blank Chrome window opened automatically in the meantime informing about Chrome being controlled by automated software.



h. The password.

After choosing the user number, there will be password window activated as following. Please put your JD Edwards password in. The password will be stored for whole time while the program is on.



i. Welcome screen.

After password is confirmed by clicking OK button, there is Welcome screen as shown below.

At the top of the screen there are branch codes displayed according to the company most commonly used. These codes can be modified via menu and option 4.

```
-----
Branch Code 1:  #1
Branch Code 2:  #2
-----

Please choose option you would like to run this time or press 0 for EXIT:

1. Login into JDE
2. Export control - data retrieve from JDE
3. Unit cost - data retrieve from JDE
4. Branch code change
0. EXIT
```

j. After choosing option 2. or 3. which are main functions of the program there will be question displayed in the terminal asking what is first and last row number the program is expected to find the data for. The row numbers refer to the corresponding Excel file rows in the *RMA requests* file.

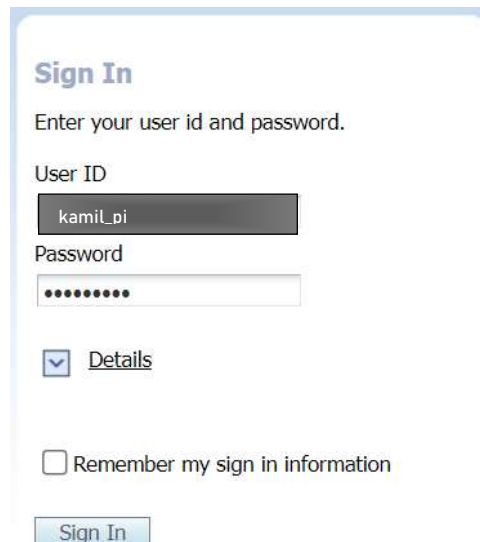
```
Enter the first 'RMA requests' row number for searching: 5
Enter the last 'RMA requests' row number for searching: 10
```

From now on, after confirmation of each line number by pressing Enter, the rest of the work in ERP system is done automatically by Python script.

At first, there is information displayed in the terminal informing about pulling credentials from the .env file.

```
Your credential file path is: C:/Users/kamil_pi/.env
Searching for credentials in: C:/Users/kamil_pi/.env
```

Promptly afterwards on the Chrome window there is JDE web application run with the login screen as following. The program will automatically put the login and password in the right place and then will click *Sign In* button.

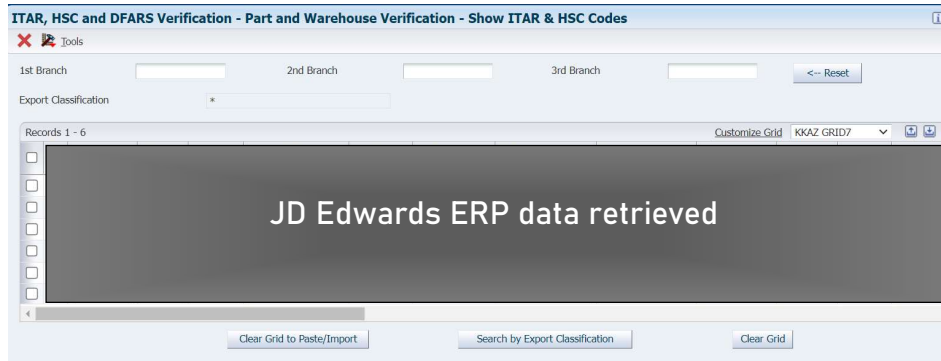
A screenshot of the JDE web application login screen. The title is "Sign In". Below it, it says "Enter your user id and password." There are two input fields: "User ID" with the value "kamil_pi" and "Password" with masked characters ".....". Below the password field is a checkbox labeled "Details" which is checked, and another checkbox labeled "Remember my sign in information" which is unchecked. At the bottom is a "Sign In" button.

After signing in there will be standard JDE application window visible for the while as shown below.



k. Menu – position 2 – Export control retrieve

Shortly afterwards the program will go into export control module and start putting the input data in. Results are generated automatically right after switching to next rows of data.



Program will also inform in the terminal:

- how many part numbers (PN) were found in the Excel file (input data),
- ERP output data was retrieved
- ERP output data was pasted into Excel

```

-----
Logging into JDE as: kamil_pi

Searching Excel file part numbers...

Part No. have been found:
[ Part Numbers from Excel file ]

Export control data retrieved from JDE

Saving data in the Excel file...

All done. Process completed

```

l. Menu - position 3 – unit price retrieve

Program will automatically start working in following loop:

Open Purchase Order Enquiry -> Paste PO -> Paste PN -> Generate data including Unit Price -> Read and store Unit price -> Open Purchase Order Enquiry ...



Subcontractor Name	Line Description	Ln Ty	2nd Item Number	Base Curr	Foreign Amount To Receive	Buyer Number	Buyer Name	Transaction Originator	Curr	Unit Cost
JD Edwards data retrieved including Unit Cost										

Once all sequences are done, program will save stored Unit prices into Excel files.

In the example below, if there are three rows for which we want to retrieve ERP data for, there will be consequently three output data fields printed out in the terminal which is shown in the picture below.

```

Logging into JDE as: username

Searching Excel file part numbers...

Part No. have been found:
[ 'Part no. #1' , 'Part no. #2' , 'Part no. #3' ]

Searching Excel file for POs and part numbers...
POs found:
[ PO#1 , PO#2 , PO#3 ]

I have found following combinations of Part No. and PO:
[( 'Part no. / PO #1' ), ( 'Part no. / PO #2' ), ( 'Part no. / PO #3' ) ]

Searching for unit cost based on PO and part number...

Searching for unit cost based on PO and part number...

Searching for unit cost based on PO and part number...

Unit cost found: [ Unit price#1 , Unit price#2 , Unit price#3 ]

All done. Process completed

```

After all loops script saves all the data extracted from ERP into *RMA Requests - script output.xlsx* file. It is up to you how to modify and use these outputs via standard Excel operations.

Raw data are available in separate tabs:

1. *Raw_JDE* – saved export control data
2. *Raw_JDE2* – saved unit price data

In the *Values_RMA_Requests* tab both export control and unit price data can be combined and presented as below.

No. P/N	PO	P/N_2	1st Branch HSC Code:	1st Branch HSC Desc:	Master HSC Code or 2nd branch HSC Code: Part description	Export classification	Unit Cost
Part Number and Purchase Order data transferred by VBA code in the same sequence as in original base file		Part Number modified to ensure all values are TEXT type	Export control data pulled from Raw_JDE tab				Unit cost data pulled from Raw_JDE2

8. Main Python functions and features

8.1 Packages and libraries

- **Os, sys, time** – essential system modules
- **Dotenv** – will read user names from .env files and can set them as environment variables

```
from dotenv import load_dotenv
```

- **Pandas** and **OpenPyXL** – libraries used to read and write Excel files

```
import pandas as pd
import openpyxl
```

- **Selenium** libraries:

- **Webdriver** – allows to manipulate web version of the ERP system

```
from selenium import webdriver

driver = webdriver.Chrome("local_path_to_chromedriver.exe_file")
```

- **WebDriverWait** – allow to wait for web elements to be loaded

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

- **expected_conditions** – work in line with WebDriverWait and allows to consider specific conditions such as presence of the web elements before further web browser handling

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

- **handling exceptions** can be done by using the following:

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

8.2 General functions

- **set_dotenv()** – function matches the user chosen from the menu (**username_choice**) with users pre-defined in env_files and sets one of these path as active for the script

```
env_files = {
    '1': 'C:/Users/user1_name/.env', '2': 'C:/Users/user2_name/.env', '3': 'C:/Users/user3_name/.env',
    '4': 'C:/Users/user4_name/.env', '5': 'C:/Users/user5_name/.env', '6': 'C:/Users/user6_name/.env',
    '7': 'C:/Users/user7_name/.env', '8': 'C:/Users/user8_name/.env'
}
```

- **get_password()** – function that allows to prompt for password and store it for the whole session while script is run

```
def get_password():
    global password_session
    password_session = pyautogui.password('Enter JD Edwards password: ')
    return password_session
```

- **login_to_jde(driver)** - login to JD Edwards (ERP)
 - uses try / except block to open the JD Edwards ERP web version
 - when it gets timeout waiting for web elements, it alternatively opens a new Chrome window and switches to that recently open one

```
def login_to_jde(driver):
    print(f"\n-----\nLogging into JDE as: ", os.getenv('user'))
    try:
        driver.get(url_jd_edwards)
    except TimeoutException:
        print("Timeout occurred while trying to open the link, opening in new window")
        driver.execute_script("window.open('url_jd_edwards')")
        driver.switch_to.window(driver.window_handles[-1])

    time.sleep(2)

    username = driver.find_element_by_id("User")
    password = driver.find_element_by_id("Password")
    username.send_keys(os.environ.get('user'))
    password.send_keys(password_session)
    click_xpath(driver, xpath_jde_sign_in) # SignIn button
```

- **click_xpath (driver, xpath)** – essential function which basically navigates to the exact web element and clicks on it

```
def click_xpath(driver, xpath):
    try:
        wait = WebDriverWait(driver, 10)
        element = wait.until(EC.presence_of_element_located((By.XPATH, xpath)))
        element.click()
    except TimeoutException:
        print("Timeout occurred while trying to open the link.")
        pass
```

8.3 Functions to get export data from JD Edwards

- **get_pn_data()** – function which opens Excel output file and reads out Part Number (PN) references that are next pasted into ERP

```
def get_pn_data():
    path = url_output_file
    sheet_name = "Values_RMA_Requests"
    global row_number1
    row_number1 = int(request_row_no1) + 1
    global row_number2
    row_number2 = int(request_row_no2) + 1

    # Read the specified range of rows from the Excel file using Pandas
    df = pd.read_excel(path, sheet_name=sheet_name, header=None, skiprows=range(row_number1 - 1),
                       usecols="B", engine="openpyxl")
    # Extract the values from the specified range of rows
    pn_data = df.iloc[:row_number2 - row_number1 + 1, 0].tolist()
    print(f"\nSearching Excel file part numbers...")
    print(f"\nPart No. have been found:\n {pn_data}")
    return pn_data
```


- **open_export_control(driver)** – function uses another previously defined **click_xpath(driver, xpath)** function customized to navigate to the export control module in JD Edwards and open it

```
def open_export_control(driver):
    while True:
        try:
            click_xpath(driver, xpath_export_control)
            break
        except:
            print("Apologies, I cannot open export control module")
```

- **retrieve_export_control()** – more complex function which pastes the data into exact fields in JD Edwards and exports the output data out of ERP to clipboard; it can be described in a few parts
 - calculating number of lines (rows) and building a list of xpaths for further navigating through ERP input fields

```
def retrieve_export_control():
    pn_data = get_pn_data()
    # Calculate the number of lines based on the difference between row_number1 and row_number2
    number_of_lines = row_number2 - row_number1 + 1
    par = number_of_lines + 1
    # Preparing list of part number field XPaths which starting from first position of part no. xpath and then rest of
    # XPaths will be added based on parametres which amount = number of lines
    list_of_xpath_pn = ["/html/body/form[3]/div/table/tbody/tr/td/div/span[10]/table/tbody/tr[2]/td/table/tbody/tr/td/"
                        "table/tbody/tr[2]/td/div/table/tbody/tr/td[2]/table/tbody/tr/td/table/tbody/tr/td[3]/div/input"]

    for p in range(2, number_of_lines + 1):
        xpath_cust_pn = f"/html/body/form[3]/div/table/tbody/tr/td/div/span[10]/table/tbody/tr[2]/td/table/tbody/tr/" \
                        f"td/table/tbody/tr[2]/td/div/table/tbody/tr/td[2]/table/tbody/tr[{p}]/td/table/tbody/tr/" \
                        f"td[3]/div/input"
        list_of_xpath_pn.append(xpath_cust_pn)
```

- switching to the iFrame which contains a table element and then writing the data into ERP fields

```
# Switch to the iframe containing the table element
iframe_element = driver.find_element_by_xpath(xpath_iframe_cust_pn)
driver.switch_to.frame(iframe_element)

time.sleep(1)

for pn, xpath_custpn, br1, br2 in zip(pn_data, list_of_xpath_pn, xpath_branch1, xpath_branch2):
    customer_pn = driver.find_element_by_xpath(xpath_custpn)
    customer_pn.send_keys(pn)

    branch1 = driver.find_element_by_xpath(br1)
    branch1.send_keys(branch1_code)

    branch2 = driver.find_element_by_xpath(br2)
    branch2.send_keys(branch2_code)
    branch2.send_keys(Keys.ENTER)
```


- ensuring all the data from ERP are generated and exporting to the clipboard

```
# xpath for an empty field below
xpath_cust_pn_param = f"/html/body/form[3]/div/table/tbody/tr/td/div/span[10]/table/tbody/tr[2]/td/table/tbody/" \
    f"tr/td/table/tbody/tr[2]/td/div/table/tbody/tr/td[2]/table/tbody/tr[{par}]/td/table/tbody/" \
    f"tr/td[3]/div/input"
custpn_empty = driver.find_element_by_xpath(xpath_cust_pn_param)

# JDE hotkey for Export Grid Data
custpn_empty.send_keys(Keys.CONTROL + Keys.SHIFT + 'E')

time.sleep(2)

data_to_clipboard = driver.find_element_by_xpath(xpath_clipboard)
data_to_clipboard.click()

apply_button = driver.find_element_by_xpath(xpath_apply)
apply_button.click()
time.sleep(.5)
apply_button.send_keys(Keys.CONTROL + 'C')
```

- **save_export_control()** – function which takes care of data segregation, writing into Excel files and saving
 - getting the data from clipboard, splitting into rows, opening Excel files and find first empty row in it

```
def save_export_control():
    # Get data from clipboard
    win32clipboard.OpenClipboard()
    clipboard_data = win32clipboard.GetClipboardData()
    win32clipboard.CloseClipboard()

    # Open Excel workbook
    url = url_output_file
    workbook = openpyxl.load_workbook(url)
    worksheet = workbook["Raw_JDE"]

    data = clipboard_data
    rows = data.split('\n')
    columns = [row.split('\t') for row in rows]

    # Find the first empty row in the "Raw_JDE" sheet
    empty_row = 1
    while worksheet.cell(row=empty_row, column=1).value is not None:
        empty_row += 1
```

- pasting the data into an empty Excel row

```
# Iterate through all rows starting from the second row
for i in range(1, len(columns)):
    # Move to the next empty row
    empty_row += 1

    # Paste the row data into the empty row
    for j, column in enumerate(columns[i], start=1):
        worksheet.cell(row=empty_row, column=j, value=str(column).strip())

time.sleep(0.5)

# Save changes
workbook.save(url)
print(f"\nSaving data in the Excel file...")
```

8.4 Functions to get unit price from JD Edwards

- **get_pn_data()** – please see already explained in 8.3
- **get_po_order()** – function which reads the Purchase Order number from the Excel file and store all numbers as a list in the *po_data* variable

```
def get_po_order():
    print(f"\nSearching Excel file for Purchase Orders...")
    path = url_output_file
    sheet_name = "Values_RMA_Requests"
    row1 = int(row_number1)
    row2 = int(row_number2)

    # Read the Excel file from the bytes object using Pandas
    df = pd.read_excel(path, sheet_name=sheet_name, header=None, skiprows=range(row1 - 1), nrows=row2 - row1 + 1,
                       usecols="C", engine="openpyxl")
    po_data = df.iloc[:, 0].tolist()
    print("\nPurchase Orders found:")
    print(po_data)
    return po_data
```

- **get_pn_and_po_combined()** – functions which prepares a list of both lists with Part Numbers and Purchase Orders before using the list in the JDE transaction

```
def get_pn_and_po_combined():
    global pn_po_combined
    pn_po_combined = list(zip(get_pn_data(), get_po_order()))
    print(f"\nI have found following combinations of Part Numbers and Purchase Orders:\n {pn_po_combined}")
    return pn_po_combined
```

- **open_po_enquiry(driver)** – function that opens JDE transaction called Purchase Order Enquiry where Unit Cost for specific combination of Part Number and Purchase Order can be found

```
def open_po_enquiry(driver):
    while True:
        try:
            driver.maximize_window()
            # PO enquiry enter
            click_xpath(driver, xpath_po)
            time.sleep(2)
            break
        except:
            print("Apologies, I cannot open PO enquiry search module. Let's try again...")
            click_xpath(driver, xpath_home_icon)
            time.sleep(2)
            click_xpath(driver, xpath_po2)
            time.sleep(1)
```

- **po_enquiry_search(pn, po)** – functions which calls another function **open_po_enquiry()** to open Purchase Order Enquiry transaction in JDE, then it switches into iframe containing necessary table elements. As while navigating through the fields there are already some asterisks (*) in there, function sends the BACKSPACE key first to remove the asterisk and then pastes the data in.

```

def po_enquiry_search(pn, po):
    # this open function has been put here because under run_unit_cost_script()
    # it did not erase previous PN input
    open_po_enquiry(driver)
    # PO input field
    switch_to_iframe(xpath_iframe_po_input, xpath_po_input)
    time.sleep(3) # time delay adjusted after many tests

    po_input = driver.find_element_by_xpath(xpath_po_input)
    po_input.send_keys(Keys.BACKSPACE)
    po_input.send_keys(po)

    item_input = driver.find_element_by_xpath(xpath_item_no_input)
    item_input.send_keys(Keys.CONTROL + 'a')
    time.sleep(.5)
    item_input.send_keys(Keys.BACKSPACE)
    item_input.send_keys(pn)

    item_input.send_keys(Keys.CONTROL + Keys.ALT + 'i')
    time.sleep(1)
    driver.switch_to.default_content()

```

- **unit_price_retrieve()** – function which is essentially a while loop which
 - looks for table element containing unit price
 - if element is not found, it scrolls the window to the right side
 - gets the text value of the unit price and stores it into *unit_cost* variable

```

def unit_price_retrieve():
    time.sleep(1)
    # switch to the iframe containing the unit price element
    iframe_element = driver.find_element_by_xpath(xpath_iframe_bottom)
    driver.switch_to.frame(iframe_element)
    # click into first empty bottom field
    bottom_field = driver.find_element_by_xpath(xpath_bottom_field[0])
    bottom_field.click()
    time.sleep(2)

    unit_price_field = None
    while unit_price_field is None:
        # try to find the element with unit price
        try:
            unit_price_field = WebDriverWait(driver, .3).until(EC.visibility_of_element_located((By.XPATH,
                                                                                                     xpath_unit_price)))
        except TimeoutException:
            print(f"Waiting too long. JDE servers might be overloaded or system error - please check if PN and PO are "
                  f"correct and try later")
            time.sleep(5)
            break
        except:
            # if the element is not found, scroll the table to the right
            records_table = driver.find_element_by_xpath(xpath_records_table)
            records_table.send_keys(Keys.ARROW_RIGHT)
            time.sleep(.3) # wait for the table to refresh

    # get the value of the table element
    try:
        unit_cost = unit_price_field.text
        driver.switch_to.default_content()
        return unit_cost
    except AttributeError:
        print("One of unit price could not be find in JDE / Purchase Order Enquiry ")
        pass

```