Project Documentation

Overview

This project is a solution that integrates the pumped oil orders from the OilMat system with workshop ERP solutions in order to free the workshop manager from the work of ensuring that the corrosponding orderline is added to the customer invoice. Components:

- A flask api to handle the incomming orders.
- A worker that consumes the orders and send them to workshop ERP solution.
- Queues that handles comunication between api and worker

In addition there will be a number of changes in both the workshop and ILX management apis to allow users to configure and handle each integration. Philip will specify these changes in a seperate doc referenced from here.

Workshop integration configuration and handling:

- Type of ERP system
- User credentials
- Active indicator
- Activate/deactivate integration

Management api:

- Get health overview for integrations
- Start/Stop integration api
- Start/Stop integration worker

The solution is build using the selenium package for python and the chrome webdriver.

Currently its planed to start a seperate flask waitressserver/api and worker pr workshop ERP integration. This ensures that problems conserning one workshop wont affect others and allow for easy scalability.

The application provides the below endpoints to interact with the workshop ERP system:

GET /alive

Checks if the API is alive and if the worker is running.

Response:

- `200 OK` with a JSON object indicating the status of API + worker and the workshop that the api instance is serving.

Response Json:

```
{
    "API is alive,
```

```
    worker running":true/false
}
```

GET /queue?type={type}

The type parameter can have the value task and error. Retrieves the current tasks in the specified queue type.

Response:

- 200 OK with a JSON object containing the list of tasks in the queue and the length of the queue.
- 400 wrong queue type

Response Json:

```
{
    "queue":"[('Gygag', '560', '10', '5', '0001', 'rolf@mandrup.dk',
'Adm@1234')]"
}
```

PUT /clear_queue

The type parameter can have the value task and error. Clears all in the specified queue type.

Response:

- 200 OK with a JSON object indicating the queue has been cleared.
- 400 wrong queue type

PUT /kill

Starts a kill thread and returns. The api will stop after 5 sec.

Response:

- 200 OK with a JSON object containg the pid of the process shutting down.

Response Json:

```
{
    {"Shutting down":99999}
}
```

PUT /start_worker

Starts the worker thread to process tasks in the queue.

Response:

- `200 OK` with a JSON object indicating the worker has started.

Response Json:

```
{
    "Worker running":true
}
```

PUT /stop_worker

Stops the worker thread.

Response:

- `200 OK` with a JSON object indicating the worker has stopped.

Response Json:

```
{
    "Worker running":false
}
```

POST /create

Adds a task to create an order line to the queue.

Request Body:

```
{
    "workshop": "Bennys Auto",
    "case_id": "558",
    "product_nr": "10",
    "product_amount": "5",
    "unique_id": "xxx",
    "username": "admin",
    "password":  "gygag",
}
```

The json should be passed as a string.

Response:

- `200 OK` with a JSON object indicating the task has been added to the queue.
- `400 Bad Request` if any required parameters are missing.

GET /check_order_status?unique_id={id}

Get the current status of a placed order, a successful order will go trough the following states:

- received
- processing
- completed

There are 2 error states:

- bad request. Set in case the body of a create request contains wrong or missing parameters.
- failed. All other error senarios.

Response:

- `200 OK` with a JSON object indicating the task has been added to the queue.
- `400` order_id missing.
- `404` order_id not found

Response Json:

```
{
    "reason":"handle varenummer","status":"failed","unique_id":"0007"
}
```

Files and Directories

requerements.txt -- all nesesary dependencies
Readme.md -- This doc
OILMAT_ERP_INTEGRATION/ -- contains the full solution
 oilmat-erp/ -- All code, head of git repo
  erp_integration_types/ -- selenium code
   au2office/
    create_erp_orderline.py
   admanager/
    create_erp_orderline.py

 flask_api/ -- api code
   app.py
   order_status_db.py

 workshop_logs/ -- One subdirectory for each integrated workshop containing queues and status db
   workshop1/
    order_status.db
    create_orderline.log
    api.log
    _task_queue/
    _error_queue/

workshop2/
workshopN/

chromedriver/

Install

Server setup

The solution is installed with the ilx-admin user in C:\OILMAT-INTEGRATION>.

The dependencies are installed in a venv virtual environment actvated by:

```
python_selenium> .\virtualenv\Scripts\activate
```

Dependencies

Install the dependencies using pip:

```
pip install -r requirements.txt
```

Start Integration

Durring normal operation an ERP integration needs to be started/restarted in the following situations:

1. A new integration is configured in gui
2. An existing integration is reconfigered in gui
3. Its specificaly requested in the ILX Systems admin gui
4. The backend server is restarted. All configured active ERP integrations must be started automaticly at server startup

Shell command to start an api instances

```
python api.py port workshop integration-type
```

Example

```
 python3 app.py 5000 'hosses' 'admanager'
```

Currently 2 integration-types exists, 'admanager' and 'au2office'