

# Workflows

Everything related Workflows, Pipelines, Modules, Schedules, Data Connections etc

- [Keywords](#)
- [Workflow Engine Python Libraries](#)
- [Finmars Standard Library](#)
- [Workflow Engine Overview](#)

# Keywords

**Workflow** - some process defined by user, it sequence of one step or many steps that do something, usually its run by some **Schedule**. Job, Pipeline are all related terms

**Workflow Engine** - a tool, Django app created by Finmars SCSA that helps to create and execute **Workflow**. This service is absolutely separated from Finmars Platform (backend) due to security and logical concerns.

**Schedule** - part of **Workflow Engine**, this service allows to execute **Workflow** on some regular schedule (with crontab) and user can define **Payload** to that Schedule as well

**Payload** - an input of your **Workflow**, normally its a JSON object (dictionary) that will be passed into Workflow Code. Its accessible via `kwargs.get("payload", None)` Payload itself is user-defined as its workflows

**Definition** - old keyword, but basically its a YAML/JSON document that describes Workflow itself (meta-file)

**Workflow** (version 1) - First Generation of Workflows, its defined in `workflow.json`. This types are still supported, but deprecated. Consider using Workflow (version 2)

Example of its structure (JSON)

```
{
  "workflow":
  {
    "user_code": "com.finmars.standard-workflow:simple-autoimport",
    "is_manager": false,
    "tasks": ["com.finmars.standard-workflow:simple-autoimport.task"]
  }
}
```

Example of its structure (YAML)

```
workflow:
  user_code: com.finmars.standard-workflow:collect-price
  is_manager: false
  tasks:
    - com.finmars.standard-workflow:collect-price.task
```

**Workflow** (version 2) - Second Generation of Workflows, its defined in workflow.json. This structure is more complex and generated by Workflow Engine itself. But user still able to tune it manually. (see Workflow Template)

Workflow Engine look at "version": property, so "version": "2", is for **Workflow** (version 2). If not defined it will be **Workflow** (version 1)

```
{
  "version": "2",
  "workflow": {
    "name": "Nested Test",
    "user_code": "com.finmars.local:nested",
    "default_payload": {
      "report_date": "2024-10-10"
    },
    "nodes": [
      {
        "id": "b33a5bc711c739b7",
        "data": {
          "node": {
            "name": "step-1",
            "type": "workflow",
            "notes": "",
            "user_code": "step-1"
          },
          "workflow": {
            "name": "Download Data",
            "user_code": "com.finmars.example-etl:download"
          },
          "source_code": "\ndef main(self, *args, **kwargs):\n    self.log(\"Hello World\")\n\n    return"
        },
        {"status": "success"}\n"
      },
      {
        "name": "step-1",
        "label": "",
        "width": 422,
        "height": 598,
        "inputs": {
          "in": {
            "id": "3f1c9322cca914ed",
            "label": "Input",
```

```
    "socket": {
      "name": "socket"
    },
    "control": null,
    "showControl": true
  },
  "payload_input": {
    "id": "caf1ac1948c12178",
    "label": "Payload",
    "socket": {
      "name": "payload_socket"
    },
    "control": null,
    "showControl": true
  }
},
"outputs": {
  "out": {
    "id": "57eee5fd73fec7e5",
    "label": "Output",
    "socket": {
      "name": "socket"
    },
    "multipleConnections": true
  }
},
"controls": {},
"position": {
  "x": -361.12865172833614,
  "y": -374.3578764111789
}
},
{
  "id": "b5744926f4976461",
  "data": {
    "node": {
      "name": "conditional-test",
      "type": "workflow",
      "notes": "",
      "user_code": "conditional-test"
```

```
    },
    "workflow": {
      "name": "Test Vault",
      "user_code": "com.finmars.local:test-vault"
    },
    "source_code": "\ndef main(self, *args, **kwargs):\n    self.log(\"Hello World\")\n\n    return\n{\n\"status\": \"success\"\n}\n"
  },
  "name": "conditional-test",
  "label": "",
  "width": 417,
  "height": 719,
  "inputs": {
    "in": {
      "id": "5cee94f3086e8e3e",
      "label": "Input",
      "socket": {
        "name": "socket"
      },
      "control": null,
      "showControl": true
    },
    "payload_input": {
      "id": "1e9f1cdb08f3b2e2",
      "label": "Payload",
      "socket": {
        "name": "payload_socket"
      },
      "control": null,
      "showControl": true
    }
  },
  "outputs": {
    "out": {
      "id": "7ea2cdd214c5ba7e",
      "label": "Output",
      "socket": {
        "name": "socket"
      },
      "multipleConnections": true
    }
  }
}
```

```

    }
  },
  "controls": {},
  "position": {
    "x": 423.5227311216107,
    "y": -488.5574374671881
  }
},
{
  "id": "4324a1babd889f5f",
  "data": {
    "node": {
      "name": "final-step",
      "type": "source_code",
      "notes": "",
      "user_code": "final-step"
    },
    "workflow": {
      "name": "Test Vault",
      "user_code": "com.finmars.local:test-vault"
    },
    "source_code": "\ndef main(self, *args, **kwargs):\n    self.log(\"Hello World\")\n\n    return"
  },
  {"status": "success"}\n"
},
  "name": "final-step",
  "label": "",
  "width": 440,
  "height": 790,
  "inputs": {
    "in": {
      "id": "969915a7f2d9fcf2",
      "label": "Input",
      "socket": {
        "name": "socket"
      },
      "control": null,
      "showControl": true
    },
    "payload_input": {
      "id": "2858c38d5fc4a37b",

```

```

        "label": "Payload",
        "socket": {
            "name": "payload_socket"
        },
        "control": null,
        "showControl": true
    }
},
"outputs": {
    "out": {
        "id": "694544a7e2761259",
        "label": "Output",
        "socket": {
            "name": "socket"
        },
        "multipleConnections": true
    }
},
"controls": {},
"position": {
    "x": 1050.2581884865385,
    "y": -536.2804491695036
}
},
{
    "id": "5de8268354a95a58",
    "data": {
        "node": {
            "name": "step-2-payload-generator",
            "type": "source_code",
            "notes": "",
            "user_code": "step-2-payload-generator"
        },
        "workflow": {},
        "source_code": "\ndef main(self, *args, **kwargs):\n    self.log(\"Hello World\")\n    \n    payload =\nkwargs.get(\"payload\")\n    \n    # options = payload.get(\"options\")\n    \n    # if self.user_code in\noptions['options_steps']:\n    #     return\n    \n    \n    # execution continue\n    return {\"hello\": \"world\"}\n"
    },
    "name": "step-2-payload-generator",
    "label": "",

```

```
"width": 316,
"height": 696,
"inputs": {
  "in": {
    "id": "1d0df286dd91ce9c",
    "label": "Input",
    "socket": {
      "name": "socket"
    },
    "control": null,
    "showControl": true
  },
  "payload_input": {
    "id": "f56cfc9b414c599d",
    "label": "Payload",
    "socket": {
      "name": "payload_socket"
    },
    "control": null,
    "showControl": true
  }
},
"outputs": {
  "out": {
    "id": "4c9d99bde3f8f940",
    "label": "Output",
    "socket": {
      "name": "socket"
    },
    "multipleConnections": true
  }
},
"controls": {},
"position": {
  "x": 1.1510722982846584,
  "y": -1111.1812663772207
}
},
"connections": [
```



```

{
  "sourceOutput": "out",
  "targetInput": "in",
  "id": "36b8b8ebc0bcd3d9",
  "source": "b33a5bc711c739b7",
  "target": "b5744926f4976461"
},
{
  "sourceOutput": "out",
  "targetInput": "in",
  "id": "9560ed0b2b5092d3",
  "source": "b5744926f4976461",
  "target": "4324a1babd889f5f"
},
{
  "sourceOutput": "out",
  "targetInput": "payload_input",
  "id": "763afddc95919939",
  "source": "5de8268354a95a58",
  "target": "b5744926f4976461"
}
],
"comments": []
}
}

```

To simplify structure it will be like:

```

{
  "version": "2",
  "workflow": {
    "name": "Nested Test",
    "user_code": "com.finmars.local:nested",
    "default_payload": {
      ... # predefined payload
    },
    "nodes": [
      ... # steps of workflow
    ]
  }
}

```

```

    ],
    "connections": [
        ... # connection between steps (defines order of execution)
    ],
    "comments": []
}
}

```

**Workflow Step** - it can be either **Task** or **Workflow** (yes, workflow could be nested)

**Task** - actual code of Workflow that will be executed, right now Workflow Engine supports only Python,

**Status** - indicator of what is going with Task/Workflow, possible statuses:

- init - task is created, but awaits its execution
- progress - task is executing
- nested-progress - same as progress, but for nested scenarios
- success - task finishes successfully
- error - something went wrong
- timeout - Workflow Engine stops a task due long execution (normally its 86400 seconds or 24hours)
- canceled - User stops a task for some reason
- wait - special status for Workflow, workflow can be set on **Pause**, so all tasks in progress will be finished and no new tasks is being executed until user **Resume** Workflow

**Parent Workflow** - so, each **Task** is part (child) of **Workflow**, that means Workflow receive status Success when all of its child Tasks will be success. Also, because Nested Workflow is also wrapped as a Task, it receives status of nested-progress instead of progress.

**Explorer** - part of Finmars Platform Services, it is a File Storage, so its used not only to store user files (like documents or invoices) but store Data from Data Providers and also is a place where Source Code of Workflows is stored. Normally it stored in path */workflows/%module\_name/%workflow\_name%/* (e.g. *workflows/com/finmars/standard-workflow/collect-prices*)

**Configuration Code** - Unique identifier of **Configuration Module**, this modules shared across Finmars **Spaces** via Finmars **Marketplace**. Its a reverse DNS pattern, e.g. *com.finmars.standard-workflow*.

**Configuration Module** - a package with some extension to Finmars Platform, normally it some Meta Files that extend configuration Platform (e.g. Transaction Types, Import Schemes) But Configuration Module also could contains Source Code of Workflows.

**Marketplace** - users able to create own Workflows that solve certain problems (e.g. integration to some Broker/Bank, or extending Finmars Pricing Engine, or even create some custom apps (e.g. PDF Builder)) and share it with other people via Marketplace. Source Code packed as Configuration Module and it can be installed later with single click

**Task** (Version 1) - example of how to register your task in Workflow (Version 1)

```
import json
import requests

from workflow.api import task


workflow_user_code="com.finmars.standard-workflow:hello-world"

@task(name="%s.task" %workflow_user_code, bind=True)
def main(self, *args, **kwargs):

    print("Hello World")
```

So you must define main function and wrap it with decorator from *from workflow.api import task*. Also you feel free to import and modules in your python code. See list of available modules [here](#).

**Workflow Template** - is a Workflow (Version 2), so instead of just using YAML/JSON structure Workflow Engine provides user with Visual Editor of Workflows

 **Finmars**  
master finances

Collapse

Home

Workflow Templates

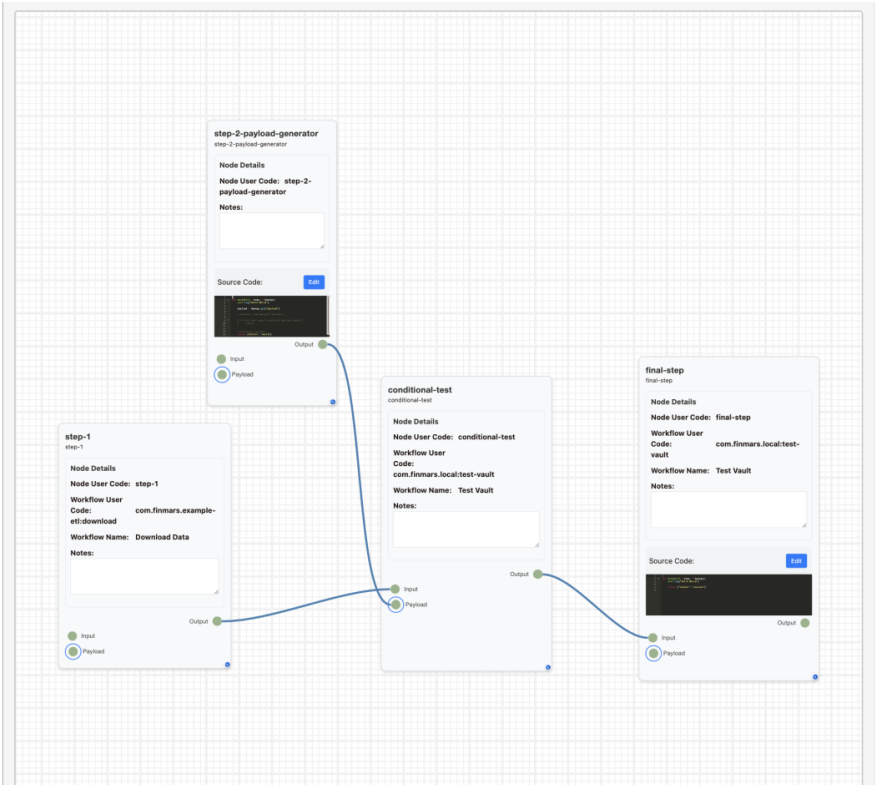
Workflows

Schedules

Documentation

API Documentation

Logs



**Workflow Template Details**

▶

📄

📁

🔍

⏪

⏩

ID

5

User Code

com.finmars.local:nested

Name

Nested Test

Notes

Daily for ...

Created

13 October 2024 at 11:45:23 CEST

Modified

17 October 2024 at 14:18:29 CEST

Default Payload

```
1 {
2   "report_date": "2024-10-10"
3 }
```

**Workflow Designer**

Node Name (Unique Step Name):

e.g., Step 1

Node User Code (Unique Step Name ASCII only):

e.g., step1

Node Notes

So user able to create new Steps and drop them on the pane and connect Steps with each other (defining order of execution)

Available Steps Types are: Workflow (Version 1), Custom Code, Workflow (Version 2)

# Workflow Engine Python Libraries

Here is complete list of libraries that available in Workflow Engine for 2024-11-03

```
amqp==5.1.1
asgiref==3.5.2
bcrypt==3.1.7
billiard==3.6.4.0
celery==5.2.7
certifi==2019.11.28
cffi==1.15.0
chardet==3.0.4
click==8.1.3
click-didyoumean==0.3.0
click-plugins==1.1.1
click-repl==0.2.0
cryptography==38.0.4
delegator.py==0.1.1
Django==4.0.6
django-celery-beat==2.6.0
django-cors-headers==3.7.0
django-filter==2.4.0
django-storages==1.13.1
django-storages-azure==1.6.8
djangorestframework==3.12.4
docutils==0.16
ecdsa==0.17.0
envoy==0.0.3
idna==3.4
importlib-metadata==1.5.0
jmespath==0.9.4
kombu==5.2.4
ndg-httpsclient==0.5.1
```

paramiko==2.11.0  
pexpect==4.8.0  
prompt-toolkit==3.0.22  
psutil==5.7.0  
psycopg2==2.9.4  
ptyprocess==0.7.0  
pyasn1==0.2.3  
pycparser==2.19  
PyNaCl==1.5.0  
pyOpenSSL==22.1.0  
pyotp==2.3.0  
python-dateutil==2.8.2  
python-jose==3.3.0  
python-keycloak-client==0.2.3  
pytz==2022.1  
requests==2.28.1  
rsa==4.8  
six==1.14.0  
sqlparse==0.3.0  
urllib3==1.26.11  
vine==5.0.0  
wcwidth==0.2.5  
websockets==8.1  
zipp==3.6.0  
kubernetes==25.3.0  
PyYAML==6.0  
gunicorn==20.0.4  
azure-core==1.26.1  
azure-storage-blob==12.14.1  
boto3==1.26.17  
botocore==1.29.17  
djangorestframework-simplejwt==5.2.2  
pluginbase==1.0.1  
jsonschema==4.17.3  
sqlalchemy==1.4.46  
flower==2.0.0  
mkdocs==1.4.2  
mkdocs-material==9.0.8  
pandas==2.2.2  
PyJWT==2.6.0

flatten\_json==0.1.13

sentry-sdk==1.28.1

openpyxl==3.1.2

suds==1.1.2

pysftp==0.2.9

rarfile==4.1

pyzipper==0.3.6

drf\_yasg==1.21.5

whitenoise==6.3.0

matplotlib==3.9.0

unidecode==1.3.8

croniter==3.0.3

# Finmars Standard Library

Part of Workflow Engine that exposes to user useful utilities to work with Workflow or Finmars Platform REST API

```
import csv
import datetime
import importlib
import json
import logging
import os
import time
from datetime import timedelta

import jwt
import pandas as pd
import requests
from django.core.files.base import ContentFile
from flatten_json import flatten

from workflow.authentication import FinmarsRefreshToken
from workflow.models import User, Space
from workflow_app import settings

_l = logging.getLogger('workflow')

class DjangoStorageHandler(logging.Handler):
    def __init__(self, log_file, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.log_file = log_file

    def emit(self, record):
        log_entry = self.format(record)

        storage = Storage()
```



```

storage.append_text(self.log_file, log_entry)

# with storage.open(self.log_file, 'a') as log_file:
#     log_file.write(log_entry + '\n')

# DEPRECATED, remove in 1.9.0
def get_access_token(ttl_minutes=60 * 8, *args, **kwargs):
    bot = User.objects.get(username="finmars_bot")

    # Define the expiration time +1 hour from now
    expiration_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=ttl_minutes)

    space = Space.objects.all().first()

    # Define the payload with the expiration time and username
    payload = {
        'username': bot.username,
        'realm_code': space.realm_code,
        'space_code': space.realm_code,
        'exp': expiration_time,
        'iat': datetime.datetime.utcnow() # Issued at time
    }

    # Encode the JWT token
    jwt_token = jwt.encode(payload, settings.SECRET_KEY, algorithm='HS256')

    token = FinmarsRefreshToken(jwt_token)

    return token

# This one is good
def get_refresh_token(ttl_minutes=60 * 8, *args, **kwargs):
    bot = User.objects.get(username="finmars_bot")

    # Define the expiration time +1 hour from now
    expiration_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=ttl_minutes)

```

```
space = Space.objects.all().first()

# Define the payload with the expiration time and username
payload = {
    'username': bot.username,
    'realm_code': space.realm_code,
    'space_code': space.realm_code,
    'exp': expiration_time,
    'iat': datetime.datetime.utcnow() # Issued at time
}

# Encode the JWT token
jwt_token = jwt.encode(payload, settings.SECRET_KEY, algorithm='HS256')

token = FinmarsRefreshToken(jwt_token)

return token


def get_domain():
    return settings.DOMAIN_NAME


def get_space():
    space = Space.objects.all().first()

    return space


def get_space_code():
    space = Space.objects.all().first()

    return space.space_code


def get_base_path():
    # TODO http or https?
    return 'https://' + get_domain() + '/' + get_realm_code() + '/' + get_space_code()
```

```

def get_realm_code():
    space = Space.objects.all().first()

    return space.realm_code


def create_logger(name, log_format=None):
    if not log_format:
        log_format = "[% (asctime)s][%(levelname)s][%(name)s][%(filename)s: %(funcName)s: %(lineno)d] - %(message)s"

    formatter = logging.Formatter(log_format)

    log_dir = "/.system/log/"

    log_file = os.path.join(log_dir, str(name) + ".log")
    file_handler = DjangoStorageHandler(log_file)
    file_handler.setFormatter(formatter)

    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)

    logger.addHandler(file_handler)

    return logger


def execute_expression(expression):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'expression': expression,
        'is_eval': True
    }

    space = get_space()

```

```

if space.realm_code and space.realm_code != 'realm00000':
    url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/utlis/expression/'
else:
    url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/utlis/expression/'

response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

if response.status_code != 200:
    raise Exception(response.text)

return response.json()

def execute_expression_procedure(payload):
    refresh = get_access_token

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
        'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/expression-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/expression-
procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()

```

```

def execute_data_procedure(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
        '/api/v1/procedures/data-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/data-
        procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()

def get_data_procedure_instance(id):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':

```

```

        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/data-procedure-instance/%s/' % id
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/data-procedure-
instance/%s/' % id

    response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()

def execute_pricing_procedure(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
        'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/procedures/pricing-procedure/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/procedures/pricing-
procedure/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()

```

```

def execute_task(task_name, payload={}):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'task_name': task_name,
        'payload': payload
    }

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/tasks/task/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/tasks/task/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def update_task_status(platform_task_id, status, result=None, error=None):
    refresh = get_refresh_token()
    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = {
        'status': status,
        'result': result,
        'error': error,
    }

```

```

url = f'{get_base_path()}/api/v1/tasks/task/{platform_task_id}/update-status/'
response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)
try:
    response.raise_for_status()
    return response.json()
except Exception as e:
    _l.error("update_task_status error: %s" % e)


def get_task(id):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
        '/api/v1/tasks/task/%s/' % id
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/tasks/task/%s/' % id

    response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

    return response.json()


def _wait_task_to_complete_recursive(task_id=None, retries=5, retry_interval=60, counter=None):
    if counter == retries:
        raise Exception("Task exceeded retries %s count" % retries)

    try:
        result = get_task(task_id)

```



```

        if result['status'] not in ['progress', 'P', 'I']:
            return result
    except Exception as e:
        _l.error("_wait_task_to_complete_recursive %s" % e)

    counter = counter + 1

    time.sleep(retry_interval)

    return _wait_task_to_complete_recursive(task_id=task_id, retries=retries, retry_interval=retry_interval,
                                           counter=counter)

def wait_task_to_complete(task_id=None, retries=5, retry_interval=60):
    counter = 0
    result = None

    result = _wait_task_to_complete_recursive(task_id=task_id, retries=retries, retry_interval=retry_interval,
                                           counter=counter)

    return result

def poll_workflow_status(workflow_id, max_retries=100, wait_time=5):
    url = f'/workflow/api/workflow/{workflow_id}/' # Replace with your actual API endpoint

    for attempt in range(max_retries):
        data = request_api(url)

        if data:
            status = data.get('status')
            _l.info(f'Attempt {attempt + 1}: Workflow status is {status}')

            if status in ['success', 'error']:
                return status # Return the status when it's success or error

        else:
            _l.error(f'Error fetching status')

```

```
time.sleep(wait_time) # Wait before the next attempt
```

```
_l.info('Max retries reached. Workflow status not successful.')
```

```
return None # Indicate that the status was not found
```

```
def _wait_procedure_to_complete_recursive(procedure_instance_id=None, retries=5, retry_interval=60,
counter=None):
```

```
    if counter == retries:
```

```
        raise Exception("Task exceeded retries %s count" % retries)
```

```
    result = get_data_procedure_instance(procedure_instance_id)
```

```
    counter = counter + 1
```

```
    if result['status'] not in ['progress', 'P', 'I']:
```

```
        return result
```

```
    time.sleep(retry_interval)
```

```
    return _wait_procedure_to_complete_recursive(procedure_instance_id=procedure_instance_id, retries=retries,
retry_interval=retry_interval, counter=counter)
```

```
def wait_procedure_to_complete(procedure_instance_id=None, retries=5, retry_interval=60):
```

```
    counter = 0
```

```
    result = None
```

```
    result = _wait_procedure_to_complete_recursive(procedure_instance_id=procedure_instance_id,
retries=retries,
```

```
retry_interval=retry_interval, counter=counter)
```

```
    return result
```

```
def execute_transaction_import(payload):
```

```
    refresh = get_refresh_token()
```

```
    # _l.info('refresh %s' % refresh.access_token)
```

```

headers = {'Content-type': 'application/json', 'Accept': 'application/json',
           'Authorization': f'Bearer {refresh.access_token}'}
data = payload

space = get_space()

if space.realm_code and space.realm_code != 'realm00000':
    url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/import/transaction-import/execute/'
else:
    url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/import/transaction-
import/execute/'

response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

if response.status_code != 200:
    raise Exception(response.text)

return response.json()

def execute_simple_import(payload):
    refresh = get_refresh_token()

    # _l.info('refresh %s' % refresh.access_token)

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    data = payload

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
'/api/v1/import/simple-import/execute/'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + '/api/v1/import/simple-import/execute/'

    response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)

```

```
if response.status_code != 200:  
    raise Exception(response.text)
```

```
return response.json()
```

```
def request_api(path, method='get', data=None):
```

```
    refresh = get_refresh_token()
```

```
    headers = {'Content-type': 'application/json', 'Accept': 'application/json',  
              'Authorization': f'Bearer {refresh.access_token}'}  

```

```
    space = get_space()
```

```
    if space.realm_code and space.realm_code != 'realm00000':
```

```
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code + path
```

```
    else:
```

```
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + path
```

```
    response = None
```

```
    if method.lower() == 'get':
```

```
        response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)
```

```
    elif method.lower() == 'post':
```

```
        response = requests.post(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)
```

```
    elif method.lower() == 'put':
```

```
        response = requests.put(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)
```

```
    elif method.lower() == 'patch':
```

```
        response = requests.patch(url=url, data=json.dumps(data), headers=headers, verify=settings.VERIFY_SSL)
```

```
    elif method.lower() == 'delete':
```

```
response = requests.delete(url=url, headers=headers, verify=settings.VERIFY_SSL)
```

```
if response.status_code != 200 and response.status_code != 201 and response.status_code != 204:  
    raise Exception(response.text)
```

```
if response.status_code != 204:  
    return response.json()
```

```
return {"status": "no_content"}
```

```
class Storage():
```

```
    def __init__(self):
```

```
        from workflow.storage import get_storage
```

```
        self.storage = get_storage()
```

```
    def get_base_path(self):
```

```
        space = Space.objects.all().first()
```

```
        return space.space_code
```

```
    def listdir(self, path):
```

```
        return self.storage.listdir('/') + self.get_base_path() + path)
```

```
    def open(self, name, mode='rb'):
```

```
        # TODO permission check
```

```
        if name[0] == '/':
```

```
            name = self.get_base_path() + name
```

```
        else:
```

```
            name = self.get_base_path() + '/' + name
```

```
        return self.storage.open(name, mode)
```

```
    def read_json(self, filepath, mode='rb'):
```

```

with self.open(filepath, mode) as state:
    state_content = json.loads(state.read())
return state_content

def read_csv(self, filepath, mode='rb'):
    with self.open(filepath, mode) as f:
        reader = csv.DictReader(f)
        data = list(reader)
    return data

def read(self, filepath, mode='rb'):

    # Open the file from your storage backend
    file_obj = self.open(filepath, mode) # 'rb' is to read in binary mode

    try:
        # Read the file's contents
        file_content = file_obj.read()
        return file_content
    finally:
        # Make sure we close the file object
        file_obj.close()

def delete(self, name):

    # TODO permission check

    if name[0] == '/':
        name = self.get_base_path() + name
    else:
        name = self.get_base_path() + '/' + name

    return self.storage.delete(name)

def exists(self, name):

    # TODO permission check

    if name[0] == '/':
        name = self.get_base_path() + name

```

```
else:
```

```
    name = self.get_base_path() + '/' + name
```

```
    return self.storage.exists(name)
```

```
def save(self, name, content):
```

```
    if name[0] == '/':
```

```
        name = self.get_base_path() + name
```

```
    else:
```

```
        name = self.get_base_path() + '/' + name
```

```
    return self.storage.save(name, content)
```

```
def save_text(self, name, content):
```

```
    if name[0] == '/':
```

```
        name = self.get_base_path() + name
```

```
    else:
```

```
        name = self.get_base_path() + '/' + name
```

```
    return self.storage.save(name, ContentFile(content.encode('utf-8')))
```

```
def append_text(self, name, content):
```

```
    if self.storage.exists(name):
```

```
        with self.open(name, 'r') as file:
```

```
            content = file.read()
```

```
            content = content + content + '\n'
```

```
    return self.storage.save(name, ContentFile(content.encode('utf-8')))
```

```
class Utils():
```

```
    def get_current_space_code(self):
```

```
        space = Space.objects.all().first()
```

```
        return space.space_code
```

```
    def get_list_of_dates_between_two_dates(self, date_from, date_to, to_string=False):
```

```
result = []
format = '%Y-%m-%d'

if not isinstance(date_from, datetime.date):
    date_from = datetime.datetime.strptime(date_from, format).date()
```

```
if not isinstance(date_to, datetime.date):
    date_to = datetime.datetime.strptime(date_to, format).date()
```

```
diff = date_to - date_from
```

```
for i in range(diff.days + 1):
    day = date_from + timedelta(days=i)
    if to_string:
        result.append(str(day))
    else:
        result.append(day)
```

```
return result
```

```
def is_business_day(self, date):
    return bool(len(pd.bdate_range(date, date)))
```

```
def get_yesterday(self, ):
    today = datetime.now()
    yesterday = today - timedelta(days=1)
    return yesterday
```

```
def get_list_of_business_days_between_two_dates(self, date_from, date_to, to_string=False):
    result = []
    format = '%Y-%m-%d'
```

```
if not isinstance(date_from, datetime.date):
    date_from = datetime.datetime.strptime(date_from, format).date()
```

```
if not isinstance(date_to, datetime.date):
    date_to = datetime.datetime.strptime(date_to, format).date()
```

```
diff = date_to - date_from
```



```

for i in range(diff.days + 1):
    day = date_from + timedelta(days=i)

    if self.is_business_day(day):

        if to_string:
            result.append(str(day))
        else:
            result.append(day)

    return result

def import_from_storage(self, file_path):
    # get the directory and the filename without extension

    space = get_space()

    if file_path[0] == '/':
        file_path = os.path.join(settings.WORKFLOW_STORAGE_ROOT + '/tasks/' + space.space_code +
file_path)
    else:
        file_path = os.path.join(settings.WORKFLOW_STORAGE_ROOT + '/tasks/' + space.space_code + '/' +
file_path)

    _l.info('import_from_storage.file_path %s' % file_path)

    directory, filename = os.path.split(file_path)
    module_name, _ = os.path.splitext(filename)

    _l.info('import_from_storage.module_name %s' % module_name)
    _l.info('import_from_storage.file_path %s' % file_path)

    loader = importlib.machinery.SourceFileLoader(module_name, file_path)
    module = loader.load_module()

    # add the directory to sys.path
    # spec = importlib.util.spec_from_file_location(module_name, file_path)
    #
    # if spec is None:
    #     raise ImportError(f"Cannot import file {filename}")

```

```

#
# module = importlib.util.module_from_spec(spec)
#
# # execute the module
# spec.loader.exec_module(module)
#
# # return the module
return module

```

```
def relative_import_from_storage(self, file_path, base_path):
```

```

"""

```

```
Imports a module from a given file path, resolving the path from a specified base path.
```

```
:param file_path: Relative or absolute path to the Python file to import.
```

```
:param base_path: Base directory against which relative paths should be resolved.
```

```
:return: The imported module.
```

```

"""

```

```
# Resolve the relative file_path against the provided base directory
```

```
absolute_file_path = os.path.normpath(os.path.join(base_path, file_path))
```

```
# _l.info(f'Normalized file path: {absolute_file_path}')
```

```
# Continue with your existing logic, but use absolute_file_path instead of file_path
```

```
directory, filename = os.path.split(absolute_file_path)
```

```
module_name, _ = os.path.splitext(filename)
```

```
# _l.info(f'import_from_storage.module_name {module_name}')
```

```
# _l.info(f'import_from_storage.file_path {absolute_file_path}')
```

```
loader = importlib.machinery.SourceFileLoader(module_name, absolute_file_path)
```

```
module = loader.load_module()
```

```
# add the directory to sys.path
```

```
# spec = importlib.util.spec_from_file_location(module_name, file_path)
```

```
#
```

```
# if spec is None:
```

```
#     raise ImportError(f"Cannot import file {filename}")
```

```
#
```

```

# module = importlib.util.module_from_spec(spec)
#
# # execute the module
# spec.loader.exec_module(module)
#
# # return the module
return module

def tree_to_flat(self, data, **kwargs):

    return flatten(data, **kwargs)

# Example conversions:
# "Héllo World!"    -> "hello_world!"
# "Café.com"        -> "cafe_com"
# "Jürgen.Smith"    -> "jurgен_smith"
# "Mañana es jueves." -> "manana_es_jueves_"
# "Gérard Depardieu" -> "gerard_depardieu"
# "naïve artist"    -> "naive_artist"
# Problem here Example conversions with different accents on 'e':
# "é" -> "e"
# "è" -> "e"
# "ê" -> "e"
# "ë" -> "e"
# Example conversions:
# "école"           -> "U233cole"
# "café.com"        -> "cafeU233_com"
# "Jürgen.Smith"    -> "jU252rgen_smith"
# "élève"           -> "U233IU232ve"
# "Mañana"          -> "manU241ana"
# "Gödel"           -> "gU246del"
def convert_to_ascii(self, input_string):
    # Convert the input string to lowercase
    input_string = input_string.lower()

    # Convert spaces and dots to underscores
    modified_string = input_string.replace(' ', '_').replace('.', '_')

    # Function to convert each character
    def to_ascii_or_unicode(char):

```

```

try:
    # Try to encode the character in ASCII
    ascii_char = char.encode('ascii')
    return ascii_char.decode() # Return as string if it's a valid ASCII character
except UnicodeEncodeError:
    # If it's not an ASCII character, return its Unicode code point
    return f"U{ord(char)}"

```

```

# Apply the conversion to each character in the string
ascii_string = ''.join(to_ascii_or_unicode(c) for c in modified_string)

```

```

return ascii_string

```

```

class Vault():

```

```

    # hashicorp

```

```

    # finmars

```

```

    def get_secret(self, path, provider="finmars"):

```

```

        refresh = get_refresh_token() # TODO refactor, should be permission check

```

```

        # _l.info('refresh %s' % refresh.access_token)

```

```

        if provider == 'finmars':

```

```

            # pieces = path.split('/')

```

```

            # engine_name = pieces[0]

```

```

            # secret_path = pieces[1]

```

```

            headers = {'Content-type': 'application/json', 'Accept': 'application/json',

```

```

                        'Authorization': f'Bearer {refresh.access_token}'}

```

```

            space = get_space()

```

```

            url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
f'/api/v1/vault/vault-record/?user_code=' + path

```

```

            response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

```

```

            if response.status_code != 200:

```

```

        raise Exception(response.text)

data = response.json()

secret_data = None

for item in data['results']:

    if path == item['user_code']:
        secret_data = item['data']

if not secret_data:
    raise Exception(f"Secret is {path} not found")

return secret_data

elif provider == 'hashicorp':

    pieces = path.split('/')
    engine_name = pieces[0]
    secret_path = pieces[1]

    headers = {'Content-type': 'application/json', 'Accept': 'application/json',
               'Authorization': f'Bearer {refresh.access_token}'}

    space = get_space()

    if space.realm_code and space.realm_code != 'realm00000':
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.realm_code + '/' + space.space_code +
f'/api/v1/vault/vault-secret/get/?engine_name={engine_name}&path={secret_path}'
    else:
        url = 'https://' + settings.DOMAIN_NAME + '/' + space.space_code + f'/api/v1/vault/vault-
secret/get/?engine_name={engine_name}&path={secret_path}'

    response = requests.get(url=url, headers=headers, verify=settings.VERIFY_SSL)

    if response.status_code != 200:
        raise Exception(response.text)

return response.json()['data']['data']

```

```
else:
```

```
    raise Exception("Unknown provider %s" % provider)
```

```
storage = Storage()
```


```
utils = Utils()
```

```
vault = Vault()
```

# Workflow Engine Overview

Here will be brief introduction of Frontend of Workflow Engine

You will start from Homepage (Home), it shows list of Definitions. Actually this page is for backward compatibility. Normally you will run your workflows from **Workflow Templates** page. But on this page you able to see all Workflows that a registered (presented in Explorer File System) and you able to **Run** them. So to see new modules you need to **Refresh Storage**

 <b>Finmars</b> master finances  Collapse  Home  Workflow Templates  Workflows  Schedules  Documentation  API Documentation  Logs	Name	User Code	Is Manager	Actions
		com.finmars.bloomberg-data-license-broker:download-instruments	No	Run
		com.finmars.bloomberg-data-license-broker:download-prices	No	Run
		com.finmars.cim-api:import-positions-custom	No	Run
		com.finmars.cim-api:import-transactions-custom	No	Run
		com.finmars.cim-api:import-transactions-daily	No	Run
		com.finmars.cim-api:move-to-download	No	Run
	Download Data	com.finmars.example-etl:download	No	Run
	Import Data	com.finmars.example-etl:import	No	Run
	Transform Data	com.finmars.example-etl:transform	No	Run
		com.finmars.julius-sftp-broker:download-data	No	Run

On left side of Application you see Sidebar, you can Navigate between different pages of Workflow Engine App. Lets Proceed to **Workflow Templates Page**