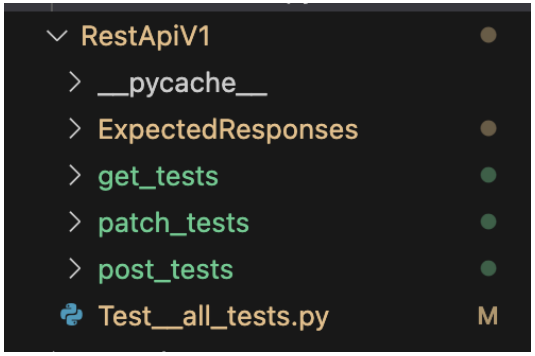
Unit Tests for DUNE-HWDB

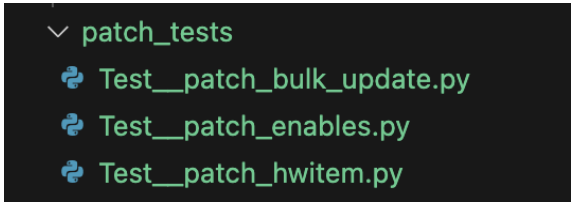
Alex Wagner
Hajime Muramatsu
Marvin Marshak
Urbas Ekka

Organization of tests

- The Unit tests are divided into 3 categories: get, post, patch.
- You can find the Unit tests in the following directory:

  /test/RestApiV1





Running tests

- Calling an individual file runs the unit tests in that particular file.

```
(base) urbas@infra04-wg012 RestApiV1 % ./Test__patch_enables.py
```

```
----------------------------------------------------------------------
Ran 2 tests in 1.896s

OK
(base) urbas@infra04-wg012 RestApiV1 % ./Test__patch_enables.py
..
----------------------------------------------------------------------
```

- All unit tests can also be run in a batch by calling ./Test__all_tests.py

```
RestApiV1 — HWDB Log — -zsh — 95×31
(base) urbas@Urbas-MacBook-Pro RestApiV1 % ./Test__all_tests.py
......ss..s.......s........s...s.s.ss
----------------------------------------------------------------------
Ran 37 tests in 6.835s

OK (skipped=9)
(base) urbas@Urbas-MacBook-Pro RestApiV1 %
```

How do they work?

- Each unit test calls a dedicated wrapper function that exists in _RestApiV1.py (/lib/Sisyphus/RestApiV1/_RestApiV1.py)
- This wrapper function is called, with required input such as part_id (and data if running a post unit test), and assigned to a variable ‘resp’
- ‘resp’ has its status and different parts of its structure checked based on what it is testing.

Example of Wrapper Function

```
def patch_bulk_enable(part_id, data, **kwargs):
    logger.debug(f"<patch_bulk_enable> part_id={part_id}")
    path = f"api/v1/components/bulk-enable"
    url = f"https://{config.rest_api}/{path}"

    resp = _patch(url, data=data, **kwargs)
    return resp
```

Example: test_get_users()

```python
def test_get_users(self):
    testname = "get_users"
    logger.info(f"[TEST {testname}]")

    try:
        resp = get_users()

        self.assertIsInstance(resp["data"][0]["user_id"],int )
        self.assertIsInstance(resp["data"][0]["username"], str)
        self.assertIsInstance(resp["data"][-1]["user_id"],int )
        self.assertIsInstance(resp["data"][-1]["username"], str)
        self.assertEqual(resp["status"], "OK")

    except AssertionError as err:
        logger.error(f"[FAIL {testname}]")
        logger.info(err)
        logger.debug(f"({testname}) response:\n{json.dumps(resp, indent=4)}")
        raise err
    logger.info(f"[PASS {testname}]")
```

Calling wrapper function

Checking that the data file retrieved has the expected structure

Example: test_post_hwitem()

```python
def test_post_hwitem(self):
    testname = "post_hwitem"
    logger.info(f"[TEST {testname}]")

    try:
        logger.info("Testing <post_component> (V1)")

        part_type_id = "Z00100300001"
        serial_number = f"SN{random.randint(0x00000000, 0xFFFFFFFF):08X}"

        data = {
            "comments": "Here are some comments",
            "component_type": {
                "part_type_id": part_type_id
            },
            "country_code": "US",
            "institution": {
                "id": 186
            },
            "manufacturer": {
                "id": 7
            },
            "serial_number": serial_number,
            "specifications":
            {
                "Widget ID": serial_number,
                "Color": "red",
                "Comment": "Unit Test: post component"
            },
            "subcomponents": {}
        }
```

```python
    resp = post_hwitem(part_type_id, data)

    logger.info(f"The response was: {resp}")

    self.assertEqual(resp["status"], "OK")

except AssertionError as err:
    logger.error(f"[FAIL {testname}]")
    logger.info(err)
    raise err

logger.info(f"[PASS {testname}]")
```

Uses wrapper function to post the part

Checks that the status of the post was ‘OK’

Example: test_patch_enable_item()

```python
def test_patch_enable_item(self):
    testname = "patch_enable_item"
    logger.info(f"[TEST {testname}]")

    try:
        #POST
        #########

        part_type_id = "Z00100300001"
        serial_number = "S99999"

        data = {
            "comments": "Here are some comments",
            "component_type": {
                "part_type_id": part_type_id
            },
            "country_code": "US",
            "institution": {
                "id": 186
            },
            "manufacturer": {
                "id": 7
            },
            "serial_number": serial_number,
            "specifications": {
                "Widget ID": serial_number,
                "Color": "red",
                "Comment": "Unit Testing"
            },
            "subcomponents": {}
        }
```

```python
logger.info(f"Posting new hwitem: part_type_id={part_type_id}, "
            f"serial_number={serial_number}")
resp = post_hwitem(part_type_id, data)
logger.info(f"Response from post: {resp}")
self.assertEqual(resp["status"], "OK")

component_id = resp["component_id"]
part_id = resp["part_id"]

logger.info(f"New hwitem result: part_id={part_id}, component_id={component_id}")
```

Wrapper function for posting item

Collecting the associated part_id and component_id from the response

Checking if the status of the response is ‘OK’

Example, continued:

```
#PATCH ENABLE
#########

data = {
    "comments": "here are some comments",
    "component": {
    "id": component_id,
    "part_id": part_id
    },
    "enabled": True,
    "geo_loc": {
    "id": 0
    }
}

resp = patch_enable_item(part_id, data)
logger.info(f"Response from patch: {resp}")
self.assertEqual(resp["status"], "OK")
#self.assertTrue(resp["enabled"])

#GET/CHECK
##########

resp = get_hwitem(part_id)
self.assertTrue(resp["data"]["enabled"])
```

Patching item based on component_id and part_id retrieved from the previous slide

Patching 'enabled' to be True (the purpose of the test)

Pushing the patched data using the wrapper function

Wrapper function to retrieve part information

Checking if the part has been enabled

Example, end:

```
        #PATCH DISABLE
        #########

        data = {
            "comments": "here are some comments",
            "component": {
            "id": component_id,
            "part_id": part_id
            },
            "enabled": False,
            "geo_loc": {
            "id": 0
            }
        }

        resp = patch_enable_item(part_id, data)
        logger.info(f"Response from patch: {resp}")
        self.assertEqual(resp["status"], "OK")

        #GET/CHECK
        ##########

        resp = get_hwitem(part_id)
        self.assertFalse(resp["data"]["enabled"])


    except AssertionError as err:
        logger.error(f"[FAIL {testname}]")
        logger.info(err)
        raise err

    logger.info(f"[PASS {testname}]")
```

Patching ‘enabled’ to be False

Pushing the patched data using the wrapper function

Checking if the part has been patched (if the part has been disabled)

Are the Unit tests ready?

- The Unit tests related to images are currently unfinished. In particular, the unit tests associated with posting images.
- Otherwise, we hope for it to be ready in the coming week/ 2 weeks.