499 lines (441 sloc) | 18.7 KB

```python
# -*- coding: utf-8 -*-
from __future__ import absolute_import

import logging
from typing import Any, Dict, List, Union

import cherrypy

from .. import DEFAULT_VERSION, mgr
from ..api.doc import Schema, SchemaInput, SchemaType
from . import ENDPOINT_MAP, BaseController, Controller, Endpoint, allow_empty_body

NO_DESCRIPTION_AVAILABLE = "*No description available*"

logger = logging.getLogger('controllers.docs')


@Controller('/docs', secure=False)
class Docs(BaseController):

    @classmethod
    def _gen_tags(cls, all_endpoints):
        """ Generates a list of all tags and corresponding descriptions. """
        # Scenarios to consider:
        #     * Intentionally make up a new tag name at controller => New tag name displayed.
        #     * Misspell or make up a new tag name at endpoint => Neither tag or endpoint displayed.
        #     * Misspell tag name at controller (when referring to another controller) =>
        #       Tag displayed but no endpoints assigned
        #     * Description for a tag added at multiple locations => Only one description displayed.
        list_of_ctrl = set()
        for endpoints in ENDPOINT_MAP.values():
            for endpoint in endpoints:
                if endpoint.is_api or all_endpoints:
                    list_of_ctrl.add(endpoint.ctrl)

        tag_map: Dict[str, str] = {}
        for ctrl in list_of_ctrl:
            tag_name = ctrl.__name__
            tag_descr = ""
            if hasattr(ctrl, 'doc_info'):
                if ctrl.doc_info['tag']:
                    tag_name = ctrl.doc_info['tag']
                tag_descr = ctrl.doc_info['tag_descr']
            if tag_name not in tag_map or not tag_map[tag_name]:
                tag_map[tag_name] = tag_descr

        tags = [{'name': k, 'description': v if v else NO_DESCRIPTION_AVAILABLE}
                for k, v in tag_map.items()]
        tags.sort(key=lambda e: e['name'])
        return tags

    @classmethod
    def _get_tag(cls, endpoint):
        """ Returns the name of a tag to assign to a path. """
        ctrl = endpoint.ctrl
        func = endpoint.func
        tag = ctrl.__name__
        if hasattr(func, 'doc_info') and func.doc_info['tag']:
            tag = func.doc_info['tag']
        elif hasattr(ctrl, 'doc_info') and ctrl.doc_info['tag']:
            tag = ctrl.doc_info['tag']
        return tag

    @classmethod
    def _gen_type(cls, param):
        # pylint: disable=too-many-return-statements
        """
        Generates the type of parameter based on its name and default value,
        using very simple heuristics.
        Used if type is not explicitly defined.
        """
        param_name = param['name']
        def_value = param['default'] if 'default' in param else None
        if param_name.startswith("is_"):
            return str(SchemaType.BOOLEAN)
        if "size" in param_name:
            return str(SchemaType.INTEGER)
        if "count" in param_name:
```

```
            return str(SchemaType.INTEGER)
        if "num" in param_name:
            return str(SchemaType.INTEGER)
        if isinstance(def_value, bool):
            return str(SchemaType.BOOLEAN)
        if isinstance(def_value, int):
            return str(SchemaType.INTEGER)
        return str(SchemaType.STRING)

    @classmethod
    # isinstance doesn't work: input is always <type 'type'>.
    def _type_to_str(cls, type_as_type):
        """ Used if type is explicitly defined. """
        if type_as_type is str:
            type_as_str = str(SchemaType.STRING)
        elif type_as_type is int:
            type_as_str = str(SchemaType.INTEGER)
        elif type_as_type is bool:
            type_as_str = str(SchemaType.BOOLEAN)
        elif type_as_type is list or type_as_type is tuple:
            type_as_str = str(SchemaType.ARRAY)
        elif type_as_type is float:
            type_as_str = str(SchemaType.NUMBER)
        else:
            type_as_str = str(SchemaType.OBJECT)
        return type_as_str

    @classmethod
    def _add_param_info(cls, parameters, p_info):
        # Cases to consider:
        #     * Parameter name (if not nested) misspelt in decorator => parameter not displayed
        #     * Sometimes a parameter is used for several endpoints (e.g. fs_id in CephFS).
        #       Currently, there is no possibility of reuse. Should there be?
        #       But what if there are two parameters with same name but different functionality?
        """
        Adds explicitly described information for parameters of an endpoint.

        There are two cases:
        * Either the parameter in p_info corresponds to an endpoint parameter. Implicit information
        has higher priority, so only information that doesn't already exist is added.
        * Or the parameter in p_info describes a nested parameter inside an endpoint parameter.
        In that case there is no implicit information at all so all explicitly described info needs
        to be added.
        """
        for p in p_info:
            if not p['nested']:
                for parameter in parameters:
                    if p['name'] == parameter['name']:
                        parameter['type'] = p['type']
                        parameter['description'] = p['description']
                        if 'nested_params' in p:
                            parameter['nested_params'] = cls._add_param_info([], p['nested_params'])
            else:
                nested_p = {
                    'name': p['name'],
                    'type': p['type'],
                    'description': p['description'],
                    'required': p['required'],
                }
                if 'default' in p:
                    nested_p['default'] = p['default']
                if 'nested_params' in p:
                    nested_p['nested_params'] = cls._add_param_info([], p['nested_params'])
                parameters.append(nested_p)

        return parameters

    @classmethod
    def _gen_schema_for_content(cls, params: List[Any]) -> Dict[str, Any]:
        """
        Generates information to the content-object in OpenAPI Spec.
        Used to for request body and responses.
        """
        required_params = []
        properties = {}
        schema_type = SchemaType.OBJECT
        if isinstance(params, SchemaInput):
            schema_type = params.type
            params = params.params

        for param in params:
            if param['required']:
                required_params.append(param['name'])

            props = {}
            if 'type' in param:
                props['type'] = cls._type_to_str(param['type'])
                if 'nested_params' in param:
                    if props['type'] == str(SchemaType.ARRAY):  # dict in array
                        props['items'] = cls._gen_schema_for_content(param['nested_params'])
                    else:  # dict in dict
                        props = cls._gen_schema_for_content(param['nested_params'])
                elif props['type'] == str(SchemaType.OBJECT):  # e.g. [int]
                    props['type'] = str(SchemaType.ARRAY)
                    props['items'] = {'type': cls._type_to_str(param['type'][0])}
            else:
                props['type'] = cls._gen_type(param)
            if 'description' in param:
```

```
                props['description'] = param['description']
            if 'default' in param:
                props['default'] = param['default']
            properties[param['name']] = props

        schema = Schema(schema_type=schema_type, properties=properties,
                        required=required_params)

        return schema.as_dict()

    @classmethod
    def _gen_responses(cls, method, resp_object=None):
        resp: Dict[str, Dict[str, Union[str, Any]]] = {
            '400': {
                "description": "Operation exception. Please check the "
                               "response body for details."
            },
            '401': {
                "description": "Unauthenticated access. Please login first."
            },
            '403': {
                "description": "Unauthorized access. Please check your "
                               "permissions."
            },
            '500': {
                "description": "Unexpected error. Please check the "
                               "response body for the stack trace."
            }
        }
        if method.lower() == 'get':
            resp['200'] = {'description': "OK",
                           'content': {'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION):
                                       {'type': 'object'}}}
        if method.lower() == 'post':
            resp['201'] = {'description': "Resource created.",
                           'content': {'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION):
                                       {'type': 'object'}}}
        if method.lower() == 'put':
            resp['200'] = {'description': "Resource updated.",
                           'content': {'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION):
                                       {'type': 'object'}}}
        if method.lower() == 'delete':
            resp['204'] = {'description': "Resource deleted.",
                           'content': {'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION):
                                       {'type': 'object'}}}
        if method.lower() in ['post', 'put', 'delete']:
            resp['202'] = {'description': "Operation is still executing."
                                          " Please check the task queue.",
                           'content': {'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION):
                                       {'type': 'object'}}}

        if resp_object:
            for status_code, response_body in resp_object.items():
                if status_code in resp:
                    resp[status_code].update({
                        'content': {
                            'application/vnd.ceph.api.v{}+json'.format(DEFAULT_VERSION): {
                                'schema': cls._gen_schema_for_content(response_body)}}})

        return resp

    @classmethod
    def _gen_params(cls, params, location):
        parameters = []
        for param in params:
            if 'type' in param:
                _type = cls._type_to_str(param['type'])
            else:
                _type = cls._gen_type(param)
            res = {
                'name': param['name'],
                'in': location,
                'schema': {
                    'type': _type
                },
            }
            if param.get('description'):
                res['description'] = param['description']
            if param['required']:
                res['required'] = True
            elif param['default'] is None:
                res['allowEmptyValue'] = True
            else:
                res['default'] = param['default']
            parameters.append(res)

        return parameters

    @classmethod
    def _gen_paths(cls, all_endpoints):
        # pylint: disable=R0912
        method_order = ['get', 'post', 'put', 'delete']
        paths = {}
        for path, endpoints in sorted(list(ENDPOINT_MAP.items()),
                                      key=lambda p: p[0]):
            methods = {}
            skip = False
```

```
            endpoint_list = sorted(endpoints, key=lambda e:
                                   method_order.index(e.method.lower()))
            for endpoint in endpoint_list:
                if not endpoint.is_api and not all_endpoints:
                    skip = True
                    break

                method = endpoint.method
                func = endpoint.func

                summary = ''
                resp = {}
                p_info = []
                if hasattr(func, 'doc_info'):
                    if func.doc_info['summary']:
                        summary = func.doc_info['summary']
                    resp = func.doc_info['response']
                    p_info = func.doc_info['parameters']
                params = []
                if endpoint.path_params:
                    params.extend(
                        cls._gen_params(
                            cls._add_param_info(endpoint.path_params, p_info), 'path'))
                if endpoint.query_params:
                    params.extend(
                        cls._gen_params(
                            cls._add_param_info(endpoint.query_params, p_info), 'query'))

                methods[method.lower()] = {
                    'tags': [cls._get_tag(endpoint)],
                    'description': func.__doc__,
                    'parameters': params,
                    'responses': cls._gen_responses(method, resp)
                }
                if summary:
                    methods[method.lower()]['summary'] = summary

                if method.lower() in ['post', 'put']:
                    if endpoint.body_params:
                        body_params = cls._add_param_info(endpoint.body_params, p_info)
                        methods[method.lower()]['requestBody'] = {
                            'content': {
                                'application/json': {
                                    'schema': cls._gen_schema_for_content(body_params)}}}

                    if endpoint.query_params:
                        query_params = cls._add_param_info(endpoint.query_params, p_info)
                        methods[method.lower()]['requestBody'] = {
                            'content': {
                                'application/json': {
                                    'schema': cls._gen_schema_for_content(query_params)}}}

                if endpoint.is_secure:
                    methods[method.lower()]['security'] = [{'jwt': []}]

            if not skip:
                paths[path] = methods

        return paths

    @classmethod
    def _gen_spec(cls, all_endpoints=False, base_url="", offline=False):
        if all_endpoints:
            base_url = ""

        host = cherrypy.request.base.split('://', 1)[1] if not offline else 'example.com'
        logger.debug("Host: %s", host)

        paths = cls._gen_paths(all_endpoints)

        if not base_url:
            base_url = "/"

        scheme = 'https' if offline or mgr.get_localized_module_option('ssl') else 'http'

        spec = {
            'openapi': "3.0.0",
            'info': {
                'description': "This is the official Ceph REST API",
                'version': "v1",
                'title': "Ceph REST API"
            },
            'host': host,
            'basePath': base_url,
            'servers': [{'url': "{}{}".format(
                cherrypy.request.base if not offline else '',
                base_url)}],
            'tags': cls._gen_tags(all_endpoints),
            'schemes': [scheme],
            'paths': paths,
            'components': {
                'securitySchemes': {
                    'jwt': {
                        'type': 'http',
                        'scheme': 'bearer',
                        'bearerFormat': 'JWT'
                    }
                }
```

```
            }
        }

        return spec

    @Endpoint(path="api.json", version=None)
    def api_json(self):
        return self._gen_spec(False, "/")

    @Endpoint(path="api-all.json", version=None)
    def api_all_json(self):
        return self._gen_spec(True, "/")

    def _swagger_ui_page(self, all_endpoints=False, token=None):
        base = cherrypy.request.base
        if all_endpoints:
            spec_url = "{}/docs/api-all.json".format(base)
        else:
            spec_url = "{}/docs/api.json".format(base)

        auth_header = cherrypy.request.headers.get('authorization')
        auth_cookie = cherrypy.request.cookie['token']
        jwt_token = ""
        if auth_cookie is not None:
            jwt_token = auth_cookie.value
        elif auth_header is not None:
            scheme, params = auth_header.split(' ', 1)
            if scheme.lower() == 'bearer':
                jwt_token = params
        else:
            if token is not None:
                jwt_token = token

        api_key_callback = """, onComplete: () => {{
                        ui.preauthorizeApiKey('jwt', '{}');
                    }}
        """.format(jwt_token)

        page = """
        <!DOCTYPE html>
        <html>
        <head>
            <meta charset="UTF-8">
            <meta name="referrer" content="no-referrer" />
            <link rel="stylesheet" type="text/css"
                  href="/swagger-ui.css" >
            <style>
                html
                {{
                    box-sizing: border-box;
                    overflow: -moz-scrollbars-vertical;
                    overflow-y: scroll;
                }}
                *,
                *:before,
                *:after
                {{
                    box-sizing: inherit;
                }}
                body {{
                    margin:0;
                    background: #fafafa;
                }}
            </style>
        </head>
        <body>
        <div id="swagger-ui"></div>
        <script src="/swagger-ui-bundle.js">
        </script>
        <script>
            window.onload = function() {{
                const ui = SwaggerUIBundle({{
                    url: '{}',
                    dom_id: '#swagger-ui',
                    presets: [
                        SwaggerUIBundle.presets.apis
                    ],
                    layout: "BaseLayout"
                    {}
                }})
                window.ui = ui
            }}
        </script>
        </body>
        </html>
        """.format(spec_url, api_key_callback)

        return page

    @Endpoint(json_response=False, version=None)
    def __call__(self, all_endpoints=False):
        return self._swagger_ui_page(all_endpoints)

    @Endpoint('POST', path="/", json_response=False,
              query_params="{all_endpoints}", version=None)
    @allow_empty_body
    def _with_token(self, token, all_endpoints=False):
        return self._swagger_ui_page(all_endpoints, token)
```

```


if __name__ == "__main__":
    import sys

    import yaml

    from . import generate_routes

    def fix_null_descr(obj):
        """
        A hot fix for errors caused by null description values when generating
        static documentation: better fix would be default values in source
        to be 'None' strings: however, decorator changes didn't resolve
        """
        return {k: fix_null_descr(v) for k, v in obj.items() if v is not None} \
            if isinstance(obj, dict) else obj

    generate_routes("/api")
    try:
        with open(sys.argv[1], 'w') as f:
            # pylint: disable=protected-access
            yaml.dump(
                fix_null_descr(Docs._gen_spec(all_endpoints=False, base_url="/", offline=True)),
                f)
    except IndexError:
        sys.exit("Output file name missing; correct syntax is: `cmd <file.yml>`")
    except IsADirectoryError:
        sys.exit("Specified output is a directory; correct syntax is: `cmd <file.yml>`")
```