Flask Bpmn

Sartography

Jan 04, 2023

CONTENTS

CHAPTER ONE

FEATURES

- Provides bmpn engine functionality for inclusion in a flask application.

CHAPTER TWO

REQUIREMENTS

- Python 3.7+

CHAPTER THREE

INSTALLATION

You can install *Flask Bpmn* via pip from PyPI:

```
$ pip install flask-bpmn
```

CHAPTER

FOUR

USAGE

Please see the Command-line Reference for details.

CHAPTER FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the Contributor Guide.

CHAPTER SIX

LICENSE

Distributed under the terms of the MIT license, *Flask Bpmn* is free and open source software.

CHAPTER SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Usage

8.1.1 flask-bpmn

Flask Bpmn.

```
flask-bpmn [OPTIONS]
```

Options

--version

Show the version and exit.

8.2 Reference

8.2.1 flask_bpmn

Flask Bpmn.

8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.3.2 How to request a feature

Request features on the Issue Tracker.

8.3.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run flask-bpmn
```

8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.4 Contributor Covenant Code of Conduct

8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at sartography@users.noreply.github.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.5 MIT License

8.6 API Reference

This page contains auto-generated API reference documentation[1].

8.6.1 `flask_bpmn`

Flask Bpmn.

Subpackages

`flask_bpmn.models`

__init__.

Submodules

`flask_bpmn.models.db`

Db.

Module Contents

Classes

SpiffworkflowBaseDBModel	SpiffworkflowBaseDBModel.

Functions

update_created_modified_on_create_listener(→ None)	Event listener that runs before a record is updated, and sets the create/modified field accordingly.
update_modified_on_update_listener(→ None)	Event listener that runs before a record is updated, and sets the modified field accordingly.
add_listeners(→ None)	Adds the listeners to all subclasses.

[1] Created with sphinx-autoapi

Attributes

db	
migrate	

`flask_bpmn.models.db.`**`db`**

`flask_bpmn.models.db.`**`migrate`**

`class` `flask_bpmn.models.db.`**`SpiffworkflowBaseDBModel`**

Bases: *db*

SpiffworkflowBaseDBModel.

`__abstract__ = True`

`classmethod _all_subclasses``()` → list[type[*SpiffworkflowBaseDBModel*]]

Get all subclasses of cls, descending.

So, if A is a subclass of B is a subclass of cls, this will include A and B. (Does not include cls)

`validate_enum_field``(`*key: str*, *value: Any*, *enum_variable: enum.EnumMeta*`)` → Any

Validate_enum_field.

`flask_bpmn.models.db.`**`update_created_modified_on_create_listener`**`(`*mapper: sqlalchemy.orm.mapper.Mapper*, *_connection: sqlalchemy.engine.base.Connection*, *target:* SpiffworkflowBaseDBModel`)` → None

Event listener that runs before a record is updated, and sets the create/modified field accordingly.

`flask_bpmn.models.db.`**`update_modified_on_update_listener`**`(`*mapper: sqlalchemy.orm.mapper.Mapper*, *_connection: sqlalchemy.engine.base.Connection*, *target:* SpiffworkflowBaseDBModel`)` → None

Event listener that runs before a record is updated, and sets the modified field accordingly.

`flask_bpmn.models.db.`**`add_listeners`**`()` → None

Adds the listeners to all subclasses.

This should be called after importing all subclasses

`flask_bpmn.models.group`

Group.

Module Contents

Classes

FlaskBpmnGroupModel	FlaskBpmnGroupModel.

class flask_bpmn.models.group.**FlaskBpmnGroupModel**

Bases: *flask_bpmn.models.db.SpiffworkflowBaseDBModel*

FlaskBpmnGroupModel.

__tablename__ = group

id

name

Submodules

`flask_bpmn.__main__`

Command-line interface.

Module Contents

Functions

main(→ None)	Flask Bpmn.

flask_bpmn.__main__.**main**() → None

Flask Bpmn.

8.6.2 api_error

API Error functionality.

Module Contents

Functions

set_user_sentry_context(→ None)	Set_user_sentry_context.
handle_exception(→ flask.wrappers.Response)	Handles unexpected exceptions.

Attributes

api_error_blueprint

`api_error.`**`api_error_blueprint`**

`exception` `api_error.`**`ApiError`**

Bases: `Exception`

ApiError Class to help handle exceptions.

`error_code :str`

`message :str`

`error_line :str =`

`error_type :str =`

`file_name :str =`

`line_number :int = 0`

`offset :int = 0`

`sentry_link :str | None`

`status_code :int = 400`

`tag :str =`

`task_data :dict | str | None`

`task_id :str =`

`task_name :str =`

`task_trace :dict | None`

`__str__`() → str

Instructions to print instance as a string.

`classmethod from_task`(*error_code: str*, *message: str*, *task: SpiffWorkflow.task.Task*, *status_code: int = 400*, *line_number: int = 0*, *offset: int = 0*, *error_type: str = ''*, *error_line: str = ''*, *task_trace: dict | None = None*) → *ApiError*

Constructs an API Error with details pulled from the current task.

static remove_unserializeable_from_dict(*my_dict: dict*) → dict

Removes unserializeable from dict.

static is_jsonable(*x: Any*) → bool

Attempts a json.dump on given input and returns false if it cannot.

classmethod from_task_spec(*code: str*, *message: str*, *task_spec: SpiffWorkflow.specs.base.TaskSpec*, *status_code: int = 400*) → *ApiError*

Constructs an API Error with details pulled from the current task.

classmethod from_workflow_exception(*error_code: str*, *message: str*, *exp: SpiffWorkflow.exceptions.WorkflowException*) → *ApiError*

Deals with workflow exceptions.

We catch a lot of workflow exception errors, so consolidating the error_code, and doing the best things we can with the data we have.

api_error.**set_user_sentry_context**() → None

Set_user_sentry_context.

api_error.**handle_exception**(*exception: Exception*) → flask.wrappers.Response

Handles unexpected exceptions.

PYTHON MODULE INDEX

INDEX