pydantic-gen

May 26, 2020

Code docs:

CHAPTER 1

Classes

1.1 SchemaGen

CHAPTER 2

Pydantic Schemas Generator

2.1 What this package does

This is a code generation package that converts YML definitions to Pydantic models (either python code or python objects).

2.2 What is Pydantic

Pydantic is a python library for data validation and settings management using python type annotations.

Take a look at the official example from the Pydantic docs.

2.3 Why generate schemas?

Normally you just program the schemas within your program, but there are several use cases when code generation makes a lot of sense:

- You're programming several apps that use the same schema (think an API server and client library for it)
- You're programming in more than one programming language

2.4 Getting started

2.4.1 Installation

Using pip:

```
pip install pydantic-gen
```

Using poetry:

```
poetry add pydantic-gen
```

2.4.2 Usage

First you need to create a YAML file with your desired class schema. See example.yml file.

```
from pydantic_gen import SchemaGen

generated = SchemaGen('example.yml')
```

The code is now generated and stored in *generated.code* attribute. There are two ways to use the code:

1. Save it to a file, and use the file in your program:

```
generated.to_file('example_output.py')
```

You can inspect the resulting file in the example_output.py

2. Or directly import the generated classed directly without saving:

```
generated.to_sys(module_name='generated_schemas')
```

After running *generated.to_sys(module_name='generated_schemas'* your generated code will be available for import:

```
import generated_schemas as gs

schema = gs.GeneratedSchema1(id=1)
```

2.4.3 Usage pattern

Recommended usage pattern is creating the yaml files needed for your projects and storing them in a separate repository, to achieve maximum consistency across all projects.

2.4.4 YAML-file structure

schemas - list of all schemas described

name - name of the generated class

props - list of properties of the class using python type annotation. Fields: *name* - field name, *type* - field type, *optional* - bool, if True the type will be wrapped in *Optional*, *default* - default value for the field.

config - list of config settings from Model Config of pydantic.

2.4.5 Testing

Project is fully covered by tests and uses pytest. To run:

```
pytest
```

2.4.6 Packaging Notice

This project uses the excellent poetry for packaging. Please read about it and let's all start using *pyproject.toml* files as a standard. Read more:

- PEP 518 – Specifying Minimum Build System Requirements for Python Projects
- What the heck is pyproject.toml?
- Clarifying PEP 518 (a.k.a. pyproject.toml)