gqlmod Documentation

Release 0.9.0

Jamie Bliss

Apr 08, 2021

CONTENTS:

gqlmod allows you to import GraphQL Query (`.gql`) files as modules and call the queries and mutations defined there. It will validate your queries at import time, to surface any problems as soon as possible.

gqlmod also defines mechanisms for handling different services (called providers) and different contexts with those services.

CHAPTER

ONE

USING GQLMOD

1.1 Summary

1. Install the `gqlmod` PyPI package, as well as any providers you need
2. Import `gqlmod.enable` as soon as possible (maybe in your `__main__.py` or top-level `__init__.py`)
3. Import your query file and start calling your queries.

1.2 Example

Listing 1: queries.gql

```
#~starwars~
query HeroForEpisode($ep: Episode!) {
  hero(episode: $ep) {
    name
    ... on Droid {
      primaryFunction
    }
    ... on Human {
      homePlanet
    }
  }
}
```

Listing 2: app.py

```
import gqlmod.enable  # noqa

from queries import HeroForEpisode

data = HeroForEpisode(ep='JEDI')
print(data)
```

Or, if you want that in async, just add `_async` at the end of the module name in your import (you do not need to change the name of the actual file).

Listing 3: app_async.py

```
import gqlmod.enable  # noqa
```

(continues on next page)

(continued from previous page)

```
from queries_async import HeroForEpisode

data = await HeroForEpisode(ep='JEDI')
print(data)
```

You may also add `_sync` to the import name to explicitly ask for the synchronous versions.

1.3 Writing Query Files

Query files are simply text files full of named GraphQL queries and mutations.

One addition is the provider declaration:

```
#~starwars~
```

This tells the system what provider to connect to these queries, and therfore how to actually query the service, what schema to validate against, etc.

The name of the provider should be in the provider's docs.

1.4 Query functions

The generated functions have a specific form.

Query functions only take keyword arguments, matching the variables defined in the query. Optional and arguments with defaults may naturally be omitted.

The function returns the data you asked for as a dict. If the server returns an error, it is raised. (gqlmod does not support GraphQL's partial results at this time.)

Note that wether query functions are synchronous or asynchronous is up to the provider; see its documentation.

1.5 Using different provider contexts

All installed providers are available at startup, initialized with no arguments. For most services, this will allow you to execute queries as an anonymous user. However, most applications will want to authenticate to the service. You can use *gqlmod.with_provider()* to provide this data to the provider.

gqlmod.with_provider() is a context manager, and may be nested. That is, you can globally authenticate as your app, but also in specific parts authenticate as a user.

The specific arguments will vary by provider, but usually have this basic form:

```
with gqlmod.with_provider('spam-service', token=config['TOKEN']):
    resp = spam_queries.GetMenu(amount_of_spam=None)
```

1.6 Major Providers

Here is a list of some maintained providers:

- `starwars`: Builtin! A demo provider that works on static constant data.
- `cirrus-ci`: From gqlmod-cirrusci, connects to Cirrus CI
- `github`: From gqlmod-github, connects to the GitHub v4 API

You may be able to discover a provider at this places:

- The gqlmod topic on GitHub
- Searching gqlmod on PyPI

1.7 API Reference

`gqlmod.`**`with_provider`**(*name*, ***params*)
Uses a new instance of the provider (with the given parameters) for the duration of the context.

`gqlmod.`**`enable_gql_import`**`()`
Enables importing `.gql` files.

Importing `gqlmod.enable` calls this.

CHAPTER

TWO

TOOL USAGE

In addition the module, several tools are available.

2.1 `gqlmod` command

Included in the package is a `gqlmod` tool. This provides static analysis functionality outside of your software.

2.1.1 `gqlmod check`

Checks graphql files for syntax and schema validty. Unlike importing, all findable errors are reported.

Give the list of files to check, or pass *–search* to scan the current directory (recursively).

2.2 GitHub Action

The check function is also available as a GitHub Action (with extra annotation integration).

Listing 1: .github/workflows/gqlmod.yml

```yaml
name: gqlmod check

on: push

jobs:
  check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v1
        with:
          fetch-depth: 1
      - uses: gqlmod/check-action@master
        with:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

You do need to use the `actions/checkout` action before calling `gqlmod/check-action`, and the `GITHUB_TOKEN` argument is required.

CHAPTER

THREE

WRITING A PROVIDER

Writing a provider is fairly staightforward.

1. Define a provider class
2. Add an entry point declaration

3.1 Provider Classes

A provider class is only required to be callable with a specific signature.

```
import graphql


class MyProvider:
    def __init__(self, token=None):
        self.token = token

    def query_sync(self, query, variables):
        # Do stuff here

        return graphql.ExecutionResult(
            errors=[],
            data={'spam': 'eggs'}
        )

    async def query_async(self, query, variables):
        # Do stuff here, asynchronously

        return graphql.ExecutionResult(
            errors=[],
            data={'spam': 'eggs'}
        )
```

The arguments it takes are:

- `query`: (string) The query to give to the server
- `variables`: (dict) The variables for that query

The provider should return a `graphql.ExecutionResult` as shown above.

3.2 Entry point

In order to be discoverable by gqlmod, providers must define entrypoints. Specifically, in the `graphql_providers` group under the name you want `.gql` files to use. This can take a few different forms, depending on your project. A few examples:

Listing 1: setup.cfg

```
[options.entry_points]
graphql_providers =
    starwars = gqlmod_starwars:StarWarsProvider
```

Listing 2: setup.py

```
setup(
    # ...
    entry_points={
        'graphql_providers': [
            'starwars = gqlmod_starwars:StarWarsProvider'
        ]
    },
    # ...
)
```

Listing 3: pyproject.toml

```
# This is for poetry-based projects
[tool.poetry.plugins.graphql_providers]
"starwars" = "gqlmod_starwars:StarWarsProvider'"
```

3.3 Extensions

In addition to the core querying interface, providers may influence the import process in a few different ways. These are all implemented as optional methods on the provider instance.

3.3.1 `get_schema_str()`

Providers may override the standard schema discovery mechanism by implementing `get_schema_str()`. This is useful for providers that don't have a primary service or don't allow anonymous access at all.

This method must be synchronous. An async variation is not supported.

Default behavior: Issue a GraphQL introspection query via the standard query path.

Parameters: None.

Returns: A `str` of the schema, in standard GraphQL schema language.

3.3.2 codegen_extra_kwargs()

Providers may add keyword arguments (variables) to the query call inside the generated module. These will be passed through the query pipeline back to the provider.

Default behavior: No additional variables are inserted.

Parameters:

- `graphql_ast` (positional, `graphql.language.OperationDefinitionNode`): The AST of the GraphQL query in question
- `schema` (positional, `graphql.type.GraphQLSchema`): The schema of the service

Returns: A `dict` of the names mapping to either simple values or `ast.AST` instances. (Note that the returned AST will be embedded into a right-hand expression context.)

3.4 Helpers

In order to help with common cases, gqlmod ships with several helpers

Note that many of them have additional requirements, which are encapsulated in extras.

3.4.1 httpx

Helpers for using `httpx` to build a provider.

Requires the `http` extra.

class gqlmod.helpers.httpx.**HttpxProvider**
Help build an HTTP-based provider based on httpx.

You should fill in *endpoint* and possibly override `modify_request_args()`.

build_request(*query*, *variables*)
Build the Request object.

Override to add authentication and such.

endpoint: **str**
The URL to send requests to.

timeout: **httpx.Timeout = None**
Timeout policy to use, if any.

3.4.2 types

Functions to help with typing of queries.

gqlmod.helpers.types.**annotate**(*ast*, *schema*)
Scans the AST and builds type information from the schema

gqlmod.helpers.types.**get_definition**(*node*)
Gets the AST object defininining the given node.

Like, a Variable node will point to a variable definition.

gqlmod.helpers.types.**get_schema**(*node*)
Gets the schema definition of the given ast node.

gqlmod.helpers.types.**get_type**(*node*, *, *unwrap=False*)
Gets the schema type of the given ast node.

If unwrap is true, also remove any wrapping types.

3.4.3 utils

gqlmod.helpers.utils.**unwrap_type**(*node*)
Gets the true type node from an schema node.

Returns the list of wrappers, the real type first and the outermost last

gqlmod.helpers.utils.**walk_query**(*query_ast*, *schema*)
Walks a query (by AST), generating 3-tuples of:

- the name path (Tuple[str])
- the AST node of the field in the query (graphql.language.ast.FieldNode)
- the schema node of the field (graphql.type.GraphQLField)

gqlmod.helpers.utils.**walk_variables**(*query_ast*, *schema*)
Walks the variables (by AST), generating 2-tuples of:

- the name path (Tuple[str])
- the schema node of the field (graphql.type.GraphQLField)

Note that the paths are rooted in the name of the variable, but the variable itself is not produced.

3.4.4 testing

gqlmod.providers.**_mock_provider**(*name*, *instance*)
Inserts and activates the given provider.

FOR TEST INFRASTRUCTURE ONLY.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

E

G

H

M

T

U

W