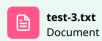
gtest-3-sv





Write a test with pytest and do not use mock's



Sure, let's write a test for the `find_app_by_string` method in `cli.py` using `pytest` without using mocks.

Here's a sample test that you can use:

```
python
 import pytest
 import ast
 from types import ModuleType
 from cli import find_app_by_string, NoAppException
 from flask import Flask
 class DummyModule:
          _init__(self, **kwargs):
         for k, v in kwargs.items():
             setattr(self, k, v)
 def test_find_app_by_string_variable():
     app = Flask( name )
     module = DummyModule(my app=app)
     result = find_app_by_string(module, 'my_app')
     assert result == app
 def test_find_app_by_string_function():
     def create_app():
         return Flask(__name__)
     module = DummyModule(create app=create app)
     result = find_app_by_string(module, 'create_app')
     assert isinstance(result, Flask)
 def test_find_app_by_string_function_with_args():
     def create_app(arg1, arg2):
         assert arg1 == 'arg1'
         assert arg2 == 'arg2'
         return Flask(__name__)
     module = DummyModule(create_app=create_app)
     result = find_app_by_string(module, 'create_app("arg1", "arg2")')
     assert isinstance(result, Flask)
 def test find app by string invalid syntax():
     module = DummyModule()
     with pytest.raises(NoAppException, match="Failed to parse"):
         find_app_by_string(module, 'invalid_syntax(')
 def test_find_app_by_string_no_such_attribute():
     module = DummyModule()
     with pytest.raises(NoAppException, match="Failed to find attribute"):
         find_app_by_string(module, 'non_existent')
```

```
def test_find_app_by_string_invalid_factory():
    def create_app(arg1):
        return Flask(__name__)

module = DummyModule(create_app=create_app)
    with pytest.raises(NoAppException, match="could not be called with the specified arguments"):
        find_app_by_string(module, 'create_app()')

def test_find_app_by_string_not_flask_instance():
    class NotFlask:
        pass

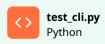
module = DummyModule(my_app=NotFlask())
    with pytest.raises(NoAppException, match="A valid Flask application was not obtained"):
    find_app_by_string(module, 'my_app')
```

Explanation:

- 2. **Creating a Dummy Module Class:** `DummyModule` is a simple class to simulate a module with attributes.
- 3. Tests:
 - `test_find_app_by_string_variable`: Tests if a variable holding a `Flask` instance is correctly found.
 - `test_find_app_by_string_function`: Tests if a function returning a `Flask` instance is correctly called and the instance is returned.
 - `test_find_app_by_string_function_with_args`: Tests if a function with arguments returning a `Flask` instance is correctly called.
 - `test_find_app_by_string_invalid_syntax`: Ensures that invalid syntax raises a `NoAppException`.
 - `test_find_app_by_string_no_such_attribute`: Ensures that a non-existent attribute raises a
 `NoAppException`.
 - `test_find_app_by_string_invalid_factory`: Ensures that calling a factory function with incorrect arguments raises a `NoAppException`.
 - `test_find_app_by_string_not_flask_instance`: Ensures that an attribute that is not a `Flask` instance raises a `NoAppException`.

Save this code in your test file and run it using `pytest`. This should cover various scenarios for the `find app by string` method.





Here is the original pytest file if it helps you write a test



Here is the content of the original `pytest` file. Let's use this as a basis to add the new test for the `find_app_by_string` method:

```
python
 import pytest
 from types import ModuleType
 from click.testing import CliRunner
 from cli import find best app, find app by string, NoAppException
 from flask import Flask
 @pytest.fixture
 def runner():
     return CliRunner()
 def test cli name(test apps):
     """Make sure the CLI object's name is the app's name and not the app itself"""
     from cliapp.app import testapp
     assert testapp.cli.name == testapp.name
 def test_find_best_app(test_apps):
     class Module:
         app = Flask("appname")
     assert find_best_app(Module) == Module.app
     class Module:
         application = Flask("appname")
     assert find best app(Module) == Module.application
     class Module:
         myapp = Flask("appname")
     assert find_best_app(Module) == Module.myapp
     class Module:
         @staticmethod
         def create_app():
             return Flask("appname")
     app = find best app(Module)
     assert isinstance(app, Flask)
     assert app.name == "appname"
     class Module:
         @staticmethod
         def create_app(**kwargs):
             return Flask("appname")
     app = find_best_app(Module)
     assert isinstance(app, Flask)
     assert app.name == "appname"
     class Module:
         @staticmethod
         def make app():
              return Flask("appname")
     app = find_best_app(Module)
     assert isinstance(app, Flask)
     assert app.name == "appname"
     class Module:
         myapp = Flask("appname1")
         @staticmethod
         def create_app():
              return Flask("appname2")
     assert find_best_app(Module) == Module.myapp
     class Module:
         myapp = Flask("appname1")
         @staticmethod
         def create_app():
              return Flask("appname2")
     assert find_best_app(Module) == Module.myapp
     class Module:
```

```
nass
    pytest.raises(NoAppException, find best app, Module)
    class Module:
        myapp1 = Flask("appname1")
        myapp2 = Flask("appname2")
    pytest.raises(NoAppException, find best app, Module)
# Adding tests for `find_app_by_string`
def test_find_app_by_string_variable():
    app = Flask(__name___)
    module = ModuleType('dummy module')
    module.my_app = app
    result = find_app_by_string(module, 'my_app')
    assert result == app
def test_find_app_by_string_function():
    def create app():
        return Flask(__name__)
    module = ModuleType('dummy_module')
    module.create_app = create_app
    result = find_app_by_string(module, 'create_app')
    assert isinstance(result, Flask)
def test_find_app_by_string_function_with_args():
    def create_app(arg1, arg2):
    assert arg1 == 'arg1'
        assert arg2 == 'arg2'
        return Flask(__name_
    module = ModuleType('dummy_module')
    module.create app = create app
    result = find_app_by_string(module, 'create_app("arg1", "arg2")')
    assert isinstance(result, Flask)
def test_find_app_by_string_invalid_syntax():
    module = ModuleType('dummy_module')
    with pytest.raises(NoAppException, match="Failed to parse"):
        find_app_by_string(module, 'invalid_syntax(')
def test_find_app_by_string_no_such_attribute():
    module = ModuleType('dummy_module')
    with pytest.raises(NoAppException, match="Failed to find attribute"):
        find app by string(module, 'non existent')
def test_find_app_by_string_invalid_factory():
    def create_app(arg1):
        return Flask(__name_
    module = ModuleType('dummy_module')
    module.create app = create app
    with pytest.raises(NoAppException, match="could not be called with the specified
arguments"):
        find_app_by_string(module, 'create_app()')
def test_find_app_by_string_not_flask_instance():
    class NotFlask:
        pass
    module = ModuleType('dummy_module')
    module.my_app = NotFlask()
    with pytest.raises(NoAppException, match="A valid Flask application was not
obtained"):
        find_app_by_string(module, 'my_app')
```

You can now add these tests to your existing `pytest` file and run them using `pytest`. These tests cover various scenarios for the `find_app_by_string` method in `cli.py`.



ChatGPT can make mistakes. Check important info.