This code creates a configurable and parameter-driven Python authentication module that supports multiple authentication methods like SPN with certificates and username-password combinations. Developers can easily integrate it into their projects by providing a configuration file or dictionary.

### **Key Features:**

1. **Pluggable Authentication Methods**:
   1. SPN with Certificate (SPNCertificateAuth).
   2. Username and Password (UserPasswordAuth).
2. **Configuration Options**:
   1. Supports JSON configuration files or direct dictionary input.
3. **Ease of Use**:
   1. The AuthConfig class abstracts the complexity of choosing and initializing the appropriate authentication method.

### **Next Steps:**

* Package this module into a Python package (setup.py) for easy installation via pip.
* Add error handling, logging improvements, and unit tests.
* Optionally support additional authentication methods, e.g., OAuth, API tokens, etc.

Auth Module Config

import os

from typing import Optional, Dict, Any

import json

import logging

from abc import ABC, abstractmethod

class AuthError(Exception):

"""Custom exception class for authentication errors."""

pass

class AuthMethod(ABC):

"""Abstract base class for authentication methods."""

@abstractmethod

def authenticate(self):

"""Abstract method to perform authentication."""

pass

class SPNCertificateAuth(AuthMethod):

"""Authentication using SPN and certificate."""

def \_\_init\_\_(self, spn: str, certificate\_path: str):

if not os.path.exists(certificate\_path):

raise AuthError(f"Certificate file not found at {certificate\_path}")

self.spn = spn

self.certificate\_path = certificate\_path

Edited Auth Module Config

### **Updates Made:**

1. **Error Handling**:
   1. Added AuthError, a custom exception class for authentication-related errors.
   2. Incorporated error handling for missing or invalid configuration and authentication inputs.
2. **Object-Oriented Features**:
   1. Encapsulated logic in classes and methods.
   2. Used abstract base classes (AuthMethod) to define a contract for authentication methods.
3. **Comments**:
   1. Added detailed comments to explain the purpose of classes, methods, and key logic for team members.

### **Next Steps:**

1. **Package the Module**:
   1. Create a setup.py for easy installation via pip.
   2. Structure the module into a directory with \_\_init\_\_.py.
2. **Extend Functionality**:
   1. Add support for additional authentication methods like OAuth or API tokens.
   2. Include unit tests for all classes and methods.
3. **Documentation**:
   1. Provide a README file with usage instructions and examples.

Setup Py

from setuptools import setup, find\_packages

setup(

name="auth\_module",

version="1.0.0",

description="A generic and configurable authentication module for Python",

author="Your Name",

[author\_email="your.email@example.com](mailto:author_email="your.email@example.com)",

packages=find\_packages(),

install\_requires=[],

classifiers=[

"Programming Language :: Python :: 3",

"License :: OSI Approved :: MIT License",

"Operating System :: OS Independent",

],

python\_requires='>=3.6',

)

### **Packaging Steps:**

1. **setup.py File**:
   1. Added a setup.py script for packaging the module.
   2. The package is named auth\_module, with metadata like version, author, and description.
2. **Directory Structure**: Ensure your project directory is structured as follows:

bash

Copy code

auth\_module/  
├── auth\_module/ # Contains your Python code files  
│ ├── \_\_init\_\_.py  
│ ├── auth\_module\_config.py # Your main code  
├── setup.py  
├── README.md # Optional, for project description

1. **Installation**: Run the following command to install the module locally:

bash

Copy code

pip install .

1. **Publishing to PyPI**:
   1. Add a README.md file for a detailed project description.
   2. Use tools like twine to upload the package to PyPI:

bash

Copy code

python setup.py sdist  
twine upload dist/\*

Readme

### **README.md Overview:**

* **Comprehensive Documentation**:
  + Describes the module's features, installation, and usage.
  + Includes examples for both configuration methods (dictionary and JSON file).
* **Error Handling**:
  + Demonstrates how to handle errors with the custom exception AuthError.
* **Extensibility**:
  + Explains how to add new authentication methods by subclassing AuthMethod.

### **Next Steps:**

* Place the README.md in the project root directory.
* Use it as the long\_description in the setup.py file for PyPI.

Readme

from setuptools import setup, find\_packages

# **Read the contents of the README file**

with open("README.md", "r", encoding="utf-8") as fh: long\_description = fh.read()

setup( name="auth\_module", version="1.0.0", description="A generic and configurable authentication module for Python", long\_description=long\_description, long\_description\_content\_type="text/markdown", author="Your Name", [author\_email="your.email@example.com](mailto:author_email="your.email@example.com)", packages=find\_packages(), install\_requires=[], classifiers=[ "Programming Language :: Python :: 3", "License :: OSI Approved :: MIT License", "Operating System :: OS Independent", ], python\_requires='>=3.6', )

Edited Readme

### **Updates to setup.py:**

* Added logic to read the README.md file as the long\_description.
* Set long\_description\_content\_type to text/markdown for proper rendering on PyPI.

### **Next Steps:**

1. Ensure the README.md file is in the project root directory.
2. Test the package installation locally:

bash

Copy code

python setup.py sdist  
pip install dist/auth\_module-1.0.0.tar.gz

1. If everything works as expected, upload the package to PyPI:

bash

Copy code

twine upload dist/\*

Let me know if you encounter any issues or need further assistance!