Versioned App using FastAPI-versioning

Versioning an API is a common practice to manage changes in your API without breaking existing clients. FastAPI allows you to easily create versioned APIs by organizing your endpoints in separate modules or routers. This allows you to provide different versions of your API, such as v1, v2, etc.

**Why Version an API?**

* **Backward Compatibility**: Ensure that changes in the API don't break existing clients.
* **New Features**: Introduce new features or changes in a new version without affecting the old one.
* **Deprecation**: Gradually phase out old features or endpoints.

**Implementing Versioning in FastAPI-Versioning**

fastapi-versioning is a library that simplifies API versioning in FastAPI applications. It provides an easy way to version your API routes without manually managing prefixes. Here’s how to use it:

**1. Install fastapi-versioning**

First, install the fastapi-versioning package using pip:

pip install fastapi-versioning

**2. Project Structure**

Let's assume you want to create two versions of your API: v1 and v2.

versionedapp/

├── main.py

├── v1/

│ ├── \_\_init\_\_.py

│ ├── endpoints.py

├── v2/

│ ├── \_\_init\_\_.py

│ ├── endpoints.py

**3. Defining Endpoints**

Define the endpoints for each version:

* **v1/endpoints.py**:

from fastapi import APIRouter

router = APIRouter()

@router.get("/books/")

def read\_books():

return [{"book\_id": 1, "title": "Book 1 (v1)"}]

@router.get("/books/{book\_id}")

def read\_book(book\_id: int):

return {"book\_id": book\_id, "title": f"Book {book\_id} (v1)"}

* **v2/endpoints.py**:

from fastapi import APIRouter

router = APIRouter()

@router.get("/books/")

def read\_books():

return [{"book\_id": 1, "title": "Book 1 (v2)"}]

@router.get("/books/{book\_id}")

def read\_book(book\_id: int):

return {"book\_id": book\_id, "title": f"Book {book\_id} (v2)"}

@router.post("/books/")

def create\_book(book: dict):

return {"book\_id": 2, "title": book["title"]}

**4. Using fastapi-versioning in main.py**

Now, let’s register these versioned routers using fastapi-versioning.

* **main.py**:

from fastapi import FastAPI

from fastapi\_versioning import VersionedFastAPI, version

from v1 import endpoints as v1\_endpoints

from v2 import endpoints as v2\_endpoints

app = FastAPI()

# Register routers with versioning

app.include\_router(v1\_endpoints.router)

app.include\_router(v2\_endpoints.router)

# Use VersionedFastAPI to handle versioning

app = VersionedFastAPI(app,

version\_format='{major}',

prefix\_format='/v{major}')

@app.get("/")

def read\_root():

return {"message": "Welcome to the API", "versions": ["v1", "v2"]}

**5. Explanation**

* **version Decorator**: While we didn’t explicitly use it in the router files, fastapi-versioning allows you to decorate individual routes with a specific version if needed. In our example, we version the entire router.
* **VersionedFastAPI**:
  + **version\_format='{major}'**: This specifies that only the major version number (e.g., v1, v2) is used in the versioning scheme.
  + **prefix\_format='/v{major}'**: This specifies how the version should appear in the URL.
* **Routing**: VersionedFastAPI automatically handles the versioning of routes based on the version decorator or the router inclusion.

**6. Accessing the Endpoints**

* **v1 API**:
  + GET /v1/books/: Returns a list of books from version 1.
  + GET /v1/books/{book\_id}: Returns details of a specific book from version 1.
* **v2 API**:
  + GET /v2/books/: Returns a list of books from version 2.
  + GET /v2/books/{book\_id}: Returns details of a specific book from version 2.
  + POST /v2/books/: Creates a new book in version 2.

**7. Optional: Specify Versions for Individual Routes**

If you want to define versions at the route level rather than the entire router, you can do so like this:

from fastapi import APIRouter

from fastapi\_versioning import version

router = APIRouter()

@router.get("/books/")

@version(1)

def read\_books\_v1():

return [{"book\_id": 1, "title": "Book 1 (v1)"}]

@router.get("/books/")

@version(2)

def read\_books\_v2():

return [{"book\_id": 1, "title": "Book 1 (v2)"}]

**Common handler across versions**

By defining a common handler outside of the versioned routers, you ensure that it is accessible regardless of the API version. This approach is useful for endpoints that should be consistent across all versions, like health checks, status endpoints, or any general-purpose routes.

**1. Define the Common Handler**

If you want to have a common handler that works across all versions of your API, you can define that handler outside of the versioned routers and include it directly in the FastAPI app. This handler will be accessible regardless of the version prefix.

**2. Example Implementation**

Let's say you have a common handler that provides some generic information about your API or a common resource like health checks.

* **main.py**:

from fastapi import FastAPI

from fastapi\_versioning import VersionedFastAPI, version

from v1 import endpoints as v1\_endpoints

from v2 import endpoints as v2\_endpoints

app = FastAPI()

# Register versioned routers

app.include\_router(v1\_endpoints.router)

app.include\_router(v2\_endpoints.router)

# Common handler not tied to any version

@app.get("/status")

def get\_status():

return {"status": "API is up and running"}

# Use VersionedFastAPI to handle versioning

app = VersionedFastAPI(app,

version\_format='{major}',

prefix\_format='/v{major}')

@app.get("/")

def read\_root():

return {"message": "Welcome to the API", "versions": ["v1", "v2"]}

**3. Accessing the Common Handler**

Since the common handler (/status) is defined outside of the versioned routers, it will be accessible across all versions without needing to specify a version prefix.

* **Common Endpoint**:
  + GET /status: This will return the status of the API, accessible regardless of the version prefix.

**4. If You Want the Common Handler to Be Versioned as Well**

If you want the common handler to be accessible with and without the version prefix, you can add it to the VersionedFastAPI configuration:

from fastapi import FastAPI

from fastapi\_versioning import VersionedFastAPI, version

from v1 import endpoints as v1\_endpoints

from v2 import endpoints as v2\_endpoints

app = FastAPI()

# Register versioned routers

app.include\_router(v1\_endpoints.router , prefix="/v1")

app.include\_router(v2\_endpoints.router , prefix="/v2")

# Common handler not tied to any version

@app.get("/status")

def get\_status():

return {"status": "API is up and running"}

# Use VersionedFastAPI to handle versioning

app = VersionedFastAPI(app,

version\_format='{major}',

prefix\_format='/v{major}',

enable\_latest=True) # Optional: Enable access to the latest version

@app.get("/")

def read\_root():

return {"message": "Welcome to the API", "versions": ["v1", "v2"]}

**5. Accessing the Common Handler with and without Versions**

* **Common Endpoint Without Version**:
  + GET /status: Accessible without versioning.
* **Common Endpoint With Version**:
  + GET /v1/status: Accessible with version 1 prefix.
  + GET /v2/status: Accessible with version 2 prefix.

**Conclusion**

fastapi-versioning simplifies the process of versioning your FastAPI applications by handling the routing and prefix management for you. This allows you to easily maintain multiple versions of your API, ensuring backward compatibility and a smooth transition between versions.