Module 7: Advanced API Design

🚀 Overview

- API versioning
- Pagination, filtering, and sorting
- Rate limiting
- Background tasks
- Caching strategies

API Versioning

Why Version APIs?

- Support backward compatibility
- Enable smooth rollout of new features
- Strategies
  - URI versioning: /v1/users/, /v2/users/
  - Query parameter: /users?version=1
  - Custom headers: Accept: application/vnd.myapi.v1+json

FastAPI Versioning Example

```python
@app.get("/v1/items/")
def get_items_v1(): ...

@app.get("/v2/items/")
def get_items_v2(): ...
```

Pagination

- **Why Use Pagination?**
  - Improve performance
  - Reduce response size
  - Enhance user experience

```
@app.get("/items/")
def list_items(skip: int = 0, limit: int = 10):
    return db[skip : skip + limit]
```

Filtering

- **Why Filter?**
  - Allow users to retrieve specific subsets of data
  - Reduce data transfer

```
@app.get("/items/")
def list_items(category: str = None):
    if category:
        return [item for item in db if item['category'] == category]
    return db
```

Sorting

- **Why Sort?**
  - Enable users to view data in a preferred order

```
@app.get("/products/")
def get_products(order_by: str = "name"):
    return sorted(db, key=lambda x: x[order_by])
```

Rate Limiting

- Why?
  - Prevent abuse
  - Protect backend resources
- Approaches
  - Fixed window (e.g., 100 req/min)
  - Sliding window
  - Token bucket

Rate Limiting with slowapi

```python
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter

@app.get("/limited/")
@limiter.limit("5/minute")
def limited_route():
    return {"message": "Limited endpoint"}
```

Background Tasks

- **Why Use Background Tasks?**
  - Offload long-running tasks
  - Improve response time for users

FastAPI Background Tasks

Example: Logging an action in the background

```
from fastapi import BackgroundTasks

def log_action(action: str):
    with open("log.txt", "a") as f:
        f.write(action)

@app.post("/action/")
def trigger_action(bg_tasks: BackgroundTasks):
    bg_tasks.add_task(log_action, "Action triggered")
    return {"status": "Task scheduled"}
```

Caching Strategies

Why Cache?

- Improve response time
- Reduce database load
- Types
  - In-memory: `functools.lru_cache`, `cachetools`
  - Distributed: Redis, Memcached
  - HTTP Caching: Cache-Control headers

FastAPI Caching Example with lrucache

- `lrucache` - a decorator for caching function results in memory.
- `lrucache` stands for "least Recently Used" cache

```
from functools import lru_cache

@lru_cache(maxsize=128)
def expensive_operation(param: str):
    return heavy_compute(param)
```

FastAPI Caching with Redis

- `aioredis` - non-blocking client for the `Redis` key-value store.
- The **aio** prefix stands for asynchronous I/O.

```
import aioredis

redis = aioredis.from_url("redis://localhost")

@app.get("/cached/")
async def get_cached_data():
    data = await redis.get("key")
    if data:
        return {"cached": True, "data": data}
```

Homework

Link to homework

Section: Practical Exercises: Advanced API Design in FastAPI

🎯 Remember

- Use versioning to manage API changes.
- Implement pagination, filtering, and sorting for better data handling.
- Apply rate limiting to protect sensitive routes.
- Use background tasks for non-blocking operations.
- Cache expensive operations to improve performance.