Week 3: FastAPI CRUD

CSI403 Full Stack Development | Lab 2 (8%)

Semester 1/2569

Agenda

1. REST API Concepts
2. FastAPI Basics
3. Pydantic Schemas
4. CRUD Operations
5. Additional Features
6. Router Organization
7. Lab 2 Assignment

What is REST API?

REpresentational **S**tate **T**ransfer

- Architectural style for web services
- Uses standard HTTP methods
- Stateless communication
- Resources identified by URLs

HTTP Methods

Method	Action	Example
GET	Read data	Get list of tasks
POST	Create data	Create new task
PUT	Update data	Update task details
DELETE	Delete data	Delete a task
PATCH	Partial update	Update task status only

HTTP Status Codes

Success (2xx):

- 200 - OK
- 201 - Created
- 204 - No Content

Client Error (4xx):

- 400 - Bad Request
- 401 - Unauthorized
- 404 - Not Found
- 422 - Validation Error

Server Error (5xx):

- 500 - Internal Server Error
- 502 - Bad Gateway
- 503 - Service Unavailable

Task API Endpoints

Method	Endpoint	Action	Status
POST	/api/tasks	Create task	201
GET	/api/tasks	List all tasks	200
GET	/api/tasks/{id}	Get one task	200
PUT	/api/tasks/{id}	Update task	200
DELETE	/api/tasks/{id}	Delete task	204

What is FastAPI?

Modern, Fast Python Web Framework

- High performance (on par with NodeJS, Go)
- Automatic API documentation (Swagger, ReDoc)
- Built-in data validation with Pydantic
- Type hints for better code quality
- Async support for better concurrency

FastAPI Hello World

```python
from fastapi import FastAPI

app = FastAPI(
    title="TaskFlow",
    description="Task Management System",
    version="1.0.0"
)

@app.get("/")
def root():
    return {"message": "Welcome to TaskFlow!"}
```

Run with: `uvicorn app.main:app -reload`

Automatic Documentation

FastAPI automatically generates API documentation:

- **Swagger UI:** http://localhost:8000/docs
- **ReDoc:** http://localhost:8000/redoc
- **OpenAPI JSON:** http://localhost:8000/openapi.json

You can test your API directly from the browser!

Path Parameters

```python
from fastapi import Path

@app.get("/tasks/{task_id}")
def get_task(task_id: int = Path(..., gt=0)):
    """
    Get a task by ID
    - task_id must be greater than 0
    """
    return {"task_id": task_id}

# Examples:
# GET /tasks/1 -> {"task_id": 1}
# GET /tasks/0 -> 422 Validation Error
# GET /tasks/abc -> 422 Validation Error
```

Query Parameters

```python
from fastapi import Query

@app.get("/tasks")
def list_tasks(
    status: str | None = Query(None),
    priority: str | None = Query(None),
    skip: int = Query(0, ge=0),
    limit: int = Query(10, ge=1, le=100)
):
    return {
        "status": status,
        "priority": priority,
        "skip": skip,
        "limit": limit
    }

# GET /tasks?status=pending&limit=5
```

What is Pydantic?

Data Validation Library

- Validates incoming data automatically
- Converts data types (e.g., string to int)
- Clear error messages for invalid data
- Generates JSON schemas
- Works seamlessly with FastAPI

Task Status Enum

```python
from enum import Enum

class TaskStatus(str, Enum):
    PENDING = "pending"
    IN_PROGRESS = "in_progress"
    DONE = "done"

class TaskPriority(str, Enum):
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"
```

Using Enum ensures only valid values are accepted.

TaskCreate Schema

```python
from pydantic import BaseModel, Field

class TaskCreate(BaseModel):
    title: str = Field(
        ...,  # Required
        min_length=1,
        max_length=200,
        description="Task title"
    )
    description: str | None = Field(
        None,
        max_length=1000
    )
    status: TaskStatus = TaskStatus.PENDING
    priority: TaskPriority = TaskPriority.MEDIUM
```

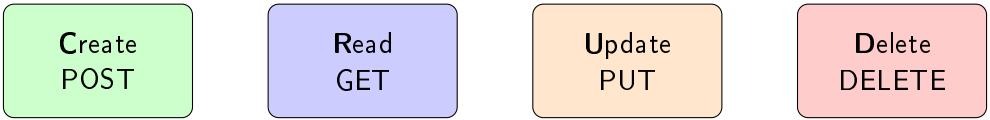
TaskResponse Schema

```python
from datetime import datetime

class TaskResponse(BaseModel):
    id: int
    title: str
    description: str | None
    status: TaskStatus
    priority: TaskPriority
    created_at: datetime
    updated_at: datetime | None

    class Config:
        from_attributes = True  # For SQLAlchemy models
```

TaskUpdate Schema

```python
class TaskUpdate(BaseModel):
    title: str | None = Field(None, min_length=1, max_length=200)
    description: str | None = None
    status: TaskStatus | None = None
    priority: TaskPriority | None = None

# All fields are optional for partial updates
# Only provided fields will be updated
```

What is CRUD?



Basic operations for any data management system.

In-Memory Storage (Temporary)

```python
from datetime import datetime

# Temporary storage (will use database later)
tasks_db: dict[int, dict] = {}
task_counter: int = 0

def get_next_id() -> int:
    global task_counter
    task_counter += 1
    return task_counter
```

Create Task (POST)

```python
from fastapi import status

@app.post("/api/tasks",
          response_model=TaskResponse,
          status_code=status.HTTP_201_CREATED)
def create_task(task: TaskCreate):
    task_id = get_next_id()
    new_task = {
        "id": task_id,
        **task.model_dump(),
        "created_at": datetime.now(),
        "updated_at": None
    }
    tasks_db[task_id] = new_task
    return new_task
```

List Tasks (GET)

```python
@app.get("/api/tasks", response_model=list[TaskResponse])
def list_tasks(
    status: TaskStatus | None = None,
    priority: TaskPriority | None = None
):
    tasks = list(tasks_db.values())

    if status:
        tasks = [t for t in tasks if t["status"] == status]
    if priority:
        tasks = [t for t in tasks if t["priority"] == priority]

    return tasks
```

Get Single Task (GET)

```python
from fastapi import HTTPException

@app.get("/api/tasks/{task_id}", response_model=TaskResponse)
def get_task(task_id: int = Path(..., gt=0)):
    if task_id not in tasks_db:
        raise HTTPException(
            status_code=404,
            detail=f"Task {task_id} not found"
        )
    return tasks_db[task_id]
```

Update Task (PUT)

```python
@app.put("/api/tasks/{task_id}", response_model=TaskResponse)
def update_task(task_id: int, task_update: TaskUpdate):
    if task_id not in tasks_db:
        raise HTTPException(status_code=404, detail="Not found")

    task = tasks_db[task_id]
    update_data = task_update.model_dump(exclude_unset=True)

    for key, value in update_data.items():
        task[key] = value

    task["updated_at"] = datetime.now()
    return task
```

Delete Task (DELETE)

```python
@app.delete("/api/tasks/{task_id}",
            status_code=status.HTTP_204_NO_CONTENT)
def delete_task(task_id: int):
    if task_id not in tasks_db:
        raise HTTPException(
            status_code=404,
            detail=f"Task {task_id} not found"
        )
    del tasks_db[task_id]
    # Returns nothing (204 No Content)
```

Search Tasks

```python
@app.get("/api/tasks/search", response_model=list[TaskResponse])
def search_tasks(q: str = Query(..., min_length=1)):
    """Search tasks by title or description"""
    results = []
    for task in tasks_db.values():
        if q.lower() in task["title"].lower():
            results.append(task)
        elif task["description"] and q.lower() in task["description"].lower():
            results.append(task)
    return results
```

Task Statistics

```python
@app.get("/api/tasks/stats")
def get_stats():
    tasks = list(tasks_db.values())
    return {
        "total": len(tasks),
        "pending": len([t for t in tasks
                        if t["status"] == "pending"]),
        "in_progress": len([t for t in tasks
                            if t["status"] == "in_progress"]),
        "done": len([t for t in tasks
                     if t["status"] == "done"])
    }
```

Using APIRouter

```python
# app/routes/tasks.py
from fastapi import APIRouter

router = APIRouter(
    prefix="/api/tasks",
    tags=["Tasks"]
)

@router.get("/")
def list_tasks():
    return []

@router.post("/")
def create_task(task: TaskCreate):
    pass
```

Include Router in Main

```python
# app/main.py
from fastapi import FastAPI
from app.routes import tasks

app = FastAPI(title="TaskFlow")

# Include the tasks router
app.include_router(tasks.router)

@app.get("/")
def root():
    return {"message": "Welcome to TaskFlow!"}
```

Lab 2: FastAPI CRUD (8%)

Requirements:

1. Create app/schemas/task.py with Pydantic schemas
2. Create app/routes/tasks.py with APIRouter
3. Implement all 5 CRUD endpoints
4. Add filtering by status and priority
5. Add search endpoint
6. Add statistics endpoint
7. Proper error handling (404, 422)

Lab 2 Grading Rubric

Criteria	Points
Pydantic Schemas (Enum, Create, Response, Update)	1.5%
CRUD Endpoints (5 endpoints)	3%
Filtering + Search + Statistics	1.5%
Error Handling (HTTPException)	1%
Code Quality + Organization	1%
Total	**8%**

Testing Your API

Use Swagger UI to test:

1. Run: `uvicorn app.main:app -reload`
2. Open: http://localhost:8000/docs
3. Click on each endpoint
4. Click "Try it out"
5. Fill in parameters
6. Click "Execute"
7. Check response

Questions?

Let's build the Task API!

Swagger: http://localhost:8000/docs