Week 4 Lab: HTTP & FastAPI

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

By the end of this lab, you will:

- Master curl, httpie, and jq for API testing
- Build a complete FastAPI application
- Handle file uploads and downloads
- Implement data validation
- Create API documentation

Structure:

- Part 1: HTTP CLI Tools (45 min)
- Part 2: FastAPI Basics (60 min)
- Part 3: File Handling (45 min)
- Part 4: Build Unit Converter API (30 min)

Setup (10 minutes)

Install tools:

```
# Install FastAPI and server
pip install fastapi uvicorn[standard] python-multipart

# Install HTTP tools (if not installed)
# macOS
brew install httpie jq

# Ubuntu/Debian
sudo apt install httpie jq

# Verify installations
http --version
jq --version
```

Exercise 1.1: Exploring APIs with curl (15 min)

Task 1: Fetch user data

```
# Get user data from JSONPlaceholder (test API)
curl https://jsonplaceholder.typicode.com/users/1

# Save to file
curl https://jsonplaceholder.typicode.com/users/1 -o user.json

# Show headers and body
curl -i https://jsonplaceholder.typicode.com/users/1
```

Task 2: Fetch all users and save

```
curl https://jsonplaceholder.typicode.com/users -o users.json
```

Exercise 1.2: curl POST Requests (15 min)

Task 1: Create a new post

```
curl -X POST https://jsonplaceholder.typicode.com/posts \
  -H "Content-Type: application/json" \
  -d '{
    "title": "My First Post",
    "body": "This is the content",
    "userId": 1
  }'
```

Task 2: Update a post (PUT)

```
curl -X PUT https://jsonplaceholder.typicode.com/posts/1 \
  -H "Content-Type: application/json" \
  -d '{
    "id": 1,
    "title": "Updated Title",
    "body": "Updated content",
```

Exercise 1.3: jq for JSON Processing (15 min)

Task 1: Extract specific fields from users

```
# Get just names
curl -s https://jsonplaceholder.typicode.com/users | jq '.[].name'

# Get name and email
curl -s https://jsonplaceholder.typicode.com/users | \
  jq '.[] | {name: .name, email: .email}'

# Users from specific city
curl -s https://jsonplaceholder.typicode.com/users | \
  jq '.[] | select(.address.city == "Gwenborough")'
```

Task 2: Build custom output

```
# Create summary
curl -s https://jsonplaceholder.typicode.com/users | \
  jq '[.[] | {username: .username, company: .company.name}]'
```

Exercise 1.4: httpie (15 min)

Task 1: Same requests, simpler syntax

```
# GET request
http https://jsonplaceholder.typicode.com/users/1

# POST request (automatic JSON)
http POST https://jsonplaceholder.typicode.com/posts \
  title="My Post" body="Content here" userId=1

# Custom headers
http https://api.github.com/users/octocat \
  Accept:application/vnd.github.v3+json
```

Task 2: Compare output

- Notice syntax highlighting
- Cleaner output format

Exercise 2.1: Hello FastAPI (15 min)

Task: Create your first API

Create `main.py`:

```
from fastapi import FastAPI

app = FastAPI(title="My First API")

@app.get("/")
def read_root():
    return {"message": "Hello World", "version": "1.0"}

@app.get("/hello/{name}")
def greet(name: str):
    return {"greeting": f"Hello, {name}!"}

@app.get("/info")
def info():
    return {
```

Exercise 2.2: Test Your API (10 min)

Task: Test all endpoints

```
# Test root
http http://localhost:8000/

# Test greeting
http http://localhost:8000/hello/Alice

# Test info
http http://localhost:8000/info

# Try the auto-generated docs
# Visit: http://localhost:8000/docs
```

Challenge:

- Add a `/status` endpoint that returns server uptime
- Add a `/math/add/{a}/{b}` endpoint

Exercise 2.3: Query Parameters (15 min)

Task: Add search and filtering

Add to `main.py`:

```
from typing import Optional

@app.get("/search")
def search(
    q: str,
    limit: int = 10,
    offset: int = 0,
    category: Optional[str] = None
):
    return {
        "query": q,
        "limit": limit,
        "offset": offset,
        "category": category,
        "results": []
    }

@app.get("/filter")
```

Exercise 2.4: Test Query Parameters (10 min)

```
# Basic search
http "http://localhost:8000/search?q=python"

# With all parameters
http "http://localhost:8000/search?q=python&limit=20&offset=10&category=books"

# Price filtering
http "http://localhost:8000/filter?min_price=10&max_price=50"
```

Challenge:

- Add validation: limit should be between 1 and 100
- Add a `/products` endpoint with filtering by category and price range

Exercise 2.5: Request Body with Pydantic (20 min)

Task: Create user registration endpoint

Add to `main.py`:

```
from pydantic import BaseModel, EmailStr, Field

class User(BaseModel):
    name: str = Field(..., min_length=1, max_length=50)
    email: EmailStr
    age: int = Field(..., ge=0, le=150)
    bio: Optional[str] = None

users_db = []

@app.post("/users", status_code=201)
def create_user(user: User):
    user_dict = user.dict()
    user_dict["id"] = len(users_db) + 1
    users_db.append(user_dict)
    return user_dict
```

Exercise 2.6: Test User Endpoints (10 min)

```
# Create valid user
http POST http://localhost:8000/users \
  name="Alice" email="alice@example.com" age=25

# Create with optional bio
http POST http://localhost:8000/users \
  name="Bob" email="bob@example.com" age=30 bio="Software Engineer"

# List all users
http http://localhost:8000/users

# Try invalid data
http POST http://localhost:8000/users \
  name="" email="invalid" age=200
```

Observe: Automatic validation errors with helpful messages

Exercise 3.1: File Upload (20 min)

Task: Implement file upload endpoint

Add to `main.py`:

```
from fastapi import File, UploadFile
import shutil
from pathlib import Path

@app.post("/upload")
async def upload_file(file: UploadFile = File(...)):
    upload_dir = Path("uploads")
    upload_dir.mkdir(exist_ok=True)

    file_path = upload_dir / file.filename

    with file_path.open("wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    return {
        "filename": file.filename,
```

Exercise 3.2: Test File Upload (10 min)

```
# Create a test file
echo "Hello, World!" > test.txt

# Upload it
http --form POST http://localhost:8000/upload \
  file@test.txt

# Upload a different file
echo "Sample CSV\nname,age\nAlice,25" > data.csv
http --form POST http://localhost:8000/upload \
  file@data.csv

# Check uploads directory
ls uploads/
```

Challenge:

- Add file size limit (e.g., max 10MB)

Exercise 3.3: File Download (15 min)

Task: Implement file download endpoint

Add to `main.py`:

```
from fastapi import HTTPException
from fastapi.responses import FileResponse

@app.get("/files")
def list_files():
    upload_dir = Path("uploads")
    if not upload_dir.exists():
        return {"files": []}

    files = [
        {
            "name": f.name,
            "size": f.stat().st_size
        }
        for f in upload_dir.iterdir() if f.is_file()
    ]
    return {"files": files}

@app.get("/download/{filename}")
def download_file(filename: str):
    file_path = Path("uploads") / filename
```

Exercise 3.4: Test File Operations (10 min)

```
# List uploaded files
http http://localhost:8000/files

# Download a file
http --download http://localhost:8000/download/test.txt

# Try downloading non-existent file
http http://localhost:8000/download/notfound.txt

# Upload multiple files and list them
echo "File 1" > file1.txt
echo "File 2" > file2.txt
http --form POST http://localhost:8000/upload file@file1.txt
http --form POST http://localhost:8000/upload file@file2.txt
http http://localhost:8000/files
```

Exercise 4: Build Unit Converter API (30 min)

Task: Complete converter with multiple units

Create `converter.py`:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from enum import Enum

app = FastAPI(title="Unit Converter API", version="1.0")

class UnitType(str, Enum):
    temperature = "temperature"
    length = "length"
    weight = "weight"

class TemperatureUnit(str, Enum):
    celsius = "celsius"
    fahrenheit = "fahrenheit"
    kelvin = "kelvin"

class LengthUnit(str, Enum):
```

Exercise 4: Converter Logic (continued)

```
class ConversionRequest(BaseModel):
    value: float
    from_unit: str
    to_unit: str
    unit_type: UnitType

# Temperature conversions
temp_conversions = {
    ("celsius", "fahrenheit"): lambda x: x * 9/5 + 32,
    ("fahrenheit", "celsius"): lambda x: (x - 32) * 5/9,
    ("celsius", "kelvin"): lambda x: x + 273.15,
    ("kelvin", "celsius"): lambda x: x - 273.15,
}

# Length conversions
length_conversions = {
    ("meter", "kilometer"): lambda x: x / 1000,
    ("kilometer", "meter"): lambda x: x * 1000,
    ("meter", "mile"): lambda x: x / 1609.34,
    ("mile", "meter"): lambda x: x * 1609.34,
    ("meter", "foot"): lambda x: x * 3.28084,
    ("foot", "meter"): lambda x: x / 3.28084,
}
```

Exercise 4: Converter Endpoint (continued)

```python
@app.post("/convert")
def convert(req: ConversionRequest):
    conversions = {}

    if req.unit_type == UnitType.temperature:
        conversions = temp_conversions
    elif req.unit_type == UnitType.length:
        conversions = length_conversions
    else:
        raise HTTPException(status_code=400, detail="Unit type not supported")

    key = (req.from_unit.lower(), req.to_unit.lower())

    if key not in conversions:
        raise HTTPException(
            status_code=400,
            detail=f"Conversion from {req.from_unit} to {req.to_unit} not supported"
        )

    result = conversions[key](req.value)

    return {
        "original_value": req.value,
        "original_unit": req.from_unit,
        "converted_value": round(result, 2),
        "converted_unit": req.to_unit
    }
```

Exercise 4: Test Converter API

```
# Run the converter API
uvicorn converter:app --reload --port 8001

# Temperature conversions
http POST http://localhost:8001/convert \
  value=0 from_unit="celsius" to_unit="fahrenheit" unit_type="temperature"

http POST http://localhost:8001/convert \
  value=100 from_unit="celsius" to_unit="kelvin" unit_type="temperature"

# Length conversions
http POST http://localhost:8001/convert \
  value=1000 from_unit="meter" to_unit="kilometer" unit_type="length"

http POST http://localhost:8001/convert \
  value=1 from_unit="mile" to_unit="kilometer" unit_type="length"
```

Exercise 4: Add Features (Bonus)

Challenge tasks:

1. **Add weight conversions**
    - kilogram, gram, pound, ounce
2. **Add history endpoint**
    - Store last 10 conversions
    - GET `/history` to retrieve
3. **Add batch conversion**
    - Convert multiple values at once
    - POST `/convert/batch`
4. **Add supported units endpoint**

Exercise 5: Error Handling (15 min)

Task: Improve error handling in your API

```
from fastapi import HTTPException, status

@app.get("/users/{user_id}")
def get_user(user_id: int):
    if user_id > len(users_db):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"User with id {user_id} not found"
        )
    return users_db[user_id - 1]

@app.delete("/users/{user_id}")
def delete_user(user_id: int):
    if user_id > len(users_db):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="User not found"
        )
```

Exercise 6: API Documentation (10 min)

Task: Enhance auto-generated docs

```python
from fastapi import FastAPI

app = FastAPI(
    title="My Awesome API",
    description="API for unit conversion and user management",
    version="1.0.0",
    contact={
        "name": "Your Name",
        "email": "your.email@example.com",
    },
)

@app.post(
    "/users",
    summary="Create a new user",
    description="Register a new user with name, email, and age",
    response_description="The created user with assigned ID",
)
def create_user(user: User):
    """
    Create a new user with the following information:

    - **name**: User's full name (1-50 characters)
    - **email**: Valid email address
    - **age**: Age between 0 and 150
```

Exercise 7: Advanced Response Models (15 min)

Task: Separate input/output models

```
class UserCreate(BaseModel):
    name: str
    email: EmailStr
    password: str
    age: int

class UserResponse(BaseModel):
    id: int
    name: str
    email: EmailStr
    age: int

@app.post("/users", response_model=UserResponse, status_code=201)
def create_user(user: UserCreate):
    # Password is in input but not in output
    user_dict = user.dict()
    user_dict["id"] = len(users_db) + 1

    # In real app: hash password before storing
    users_db.append(user_dict)
```

Exercise 8: Dependency Injection (15 min)

Task: Create reusable dependencies

```python
from fastapi import Depends, Header, HTTPException

def verify_token(x_token: str = Header(...)):
    if x_token != "secret-token":
        raise HTTPException(status_code=403, detail="Invalid token")
    return x_token

@app.get("/protected", dependencies=[Depends(verify_token)])
def protected_route():
    return {"message": "You have access!"}

@app.get("/users/me")
def get_current_user(token: str = Depends(verify_token)):
    return {
        "username": "current_user",
        "token_used": token
    }
```

Exercise 9: Background Tasks (15 min)

Task: Send notifications after user creation

```
from fastapi import BackgroundTasks
import time

def send_welcome_email(email: str, name: str):
    time.sleep(2)  # Simulate slow email sending
    print(f"Email sent to {email}: Welcome, {name}!")

@app.post("/register")
def register_user(
    user: User,
    background_tasks: BackgroundTasks
):
    users_db.append(user.dict())

    background_tasks.add_task(
        send_welcome_email,
        user.email,
        user.name
    )

    return {
        "message": "User registered. Welcome email will be sent.",
```

Putting It All Together

Your completed API should have:

1. User CRUD operations (Create, Read, Update, Delete)
2. File upload and download
3. Unit converter with multiple types
4. Query parameters and path parameters
5. Request/response validation
6. Error handling
7. API documentation
8. Background tasks

Test everything works:

Deliverables

Submit the following:

1. `main.py` - Complete API with all endpoints
2. `converter.py` - Unit converter API
3. `test_api.py` - Test file with at least 5 test cases
4. `README.md` - Brief documentation:
    - How to run the API
    - List of endpoints
    - Example requests
    - Any challenges faced

Bonus:

- Add authentication with API keys

Testing Checklist

Before submission, verify:

- [ ] All endpoints return correct status codes
- [ ] Invalid input returns 422 with error details
- [ ] File upload/download works
- [ ] Unit conversions are accurate
- [ ] API docs are accessible at `/docs`
- [ ] Error handling works (404, 400, etc.)
- [ ] Background tasks execute
- [ ] Query parameters validated
- [ ] Response models filter sensitive data

Common Issues and Solutions

Issue: Module not found

```
pip install fastapi uvicorn[standard] python-multipart
```

Issue: Port already in use

```
uvicorn main:app --reload --port 8001
```

Issue: File upload fails

- Check `python-multipart` is installed
- Use `--form` flag with httpie
- Use `-F` flag with curl

Issue: CORS errors

Resources

Official Documentation:

- FastAPI: https://fastapi.tiangolo.com/
- Pydantic: https://docs.pydantic.dev/
- Uvicorn: https://www.uvicorn.org/

Testing APIs:

- JSONPlaceholder: https://jsonplaceholder.typicode.com/
- ReqRes: https://reqres.in/
- httpbin: https://httpbin.org/

Tools:

- httpie: https://httpie.io/

Great Work!

Next week: Git fundamentals and integrating external APIs

Questions? Office hours: Tomorrow 3-5 PM