In this assignment, I'm using AI to help me through the entire project. Because I didn't know much about HTTP, REST, or JOSE, I began investigating how to get started. I asked CHATGPT(AI) "How to get started and tell me step by step, so I can understand as I go."
**It responded:**

Great assignment — this is a perfect fit for a small REST service. Since your instructor suggested using an “unfamiliar” language, I’ll implement this in **Python with FastAPI** (simple, modern, strongly REST-oriented, and easy to test).

Below is a **step-by-step build guide** with full working code, tests, and structure so you can push directly to GitHub.

---

### ✅ Step 1 — Install Dependencies

Create a project folder:

```
mkdir jwks-server
cd jwks-server
python -m venv venv
source venv/bin/activate   # Mac/Linux
venv\Scripts\activate      # Windows
```

Install packages:

```
pip install fastapi uvicorn pyjwt cryptography pytest httpx pytest-cov
```

---

### ✅ Step 2 — Key Generation Module

We generate:

- RSA key pairs
- kid
- expiry timestamp
- store active + expired keys

**keys.py**

```
import time
import uuid
from cryptography.hazmat.primitives.asymmetric import rsa

KEYS = []

def generate_rsa_keypair(expiry_seconds: int):
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )

    kid = str(uuid.uuid4())
    expiry = int(time.time()) + expiry_seconds

    key_record = {
        "kid": kid,
        "private_key": private_key,
        "public_key": private_key.public_key(),
        "expiry": expiry
    }

    KEYS.append(key_record)
    return key_record


def get_active_keys():
    now = int(time.time())
    return [k for k in KEYS if k["expiry"] > now]


def get_expired_keys():
    now = int(time.time())
    return [k for k in KEYS if k["expiry"] <= now]
```

---

### ✅ Step 3 — JWK Conversion Helper

We must convert RSA public key → JWK format.

**jwk_utils.py**

```
import base64


def b64url_uint(val: int) -> str:
    data = val.to_bytes((val.bit_length() + 7) // 8, "big")
    return base64.urlsafe_b64encode(data).rstrip(b"=").decode()


def public_key_to_jwk(public_key, kid: str):
    numbers = public_key.public_numbers()

    return {
        "kty": "RSA",
        "use": "sig",
        "alg": "RS256",
        "kid": kid,
        "n": b64url_uint(numbers.n),
        "e": b64url_uint(numbers.e),
    }
```

---

**✅ Step 4 — Main FastAPI Server**

Requirements covered:

- ✅ Port 8080
- ✅ `/jwks` endpoint
- ✅ only non-expired keys served
- ✅ `/auth` POST endpoint
- ✅ supports `?expired=true`
- ✅ JWT contains kid
- ✅ expired JWT signed with expired key

**app.py**

```
import time
import jwt
```

```
from fastapi import FastAPI, Query
from fastapi.responses import JSONResponse

from keys import generate_rsa_keypair, get_active_keys, get_expired_keys
from jwk_utils import public_key_to_jwk

app = FastAPI()

# Generate one valid + one expired key at startup
valid_key = generate_rsa_keypair(expiry_seconds=3600)
expired_key = generate_rsa_keypair(expiry_seconds=-3600)


@app.get("/jwks")
def jwks():
    active = get_active_keys()

    jwks_keys = [
        public_key_to_jwk(k["public_key"], k["kid"])
        for k in active
    ]

    return {"keys": jwks_keys}


@app.post("/auth")
def auth(expired: bool = Query(False)):
    if expired:
        key = get_expired_keys()[0]
    else:
        key = get_active_keys()[0]

    now = int(time.time())

    payload = {
        "sub": "fake-user",
        "iat": now,
        "exp": key["expiry"]
    }

    token = jwt.encode(
        payload,
```

```
        key["private_key"],
        algorithm="RS256",
        headers={"kid": key["kid"]}
    )

    return JSONResponse({"token": token})
```

---

### ✅ Step 5 — Run Server

```
uvicorn app:app --port 8080
```

Test manually:

**JWKS endpoint**

```
GET http://localhost:8080/jwks
```

**Auth endpoint**

```
POST http://localhost:8080/auth
```

Expired JWT:

```
POST http://localhost:8080/auth?expired=true
```

---

### ✅ Step 6 — Tests

We do blackbox tests using HTTP client.

**test_app.py**

```
from fastapi.testclient import TestClient
from app import app

client = TestClient(app)


def test_jwks_returns_keys():
    r = client.get("/jwks")
    assert r.status_code == 200
```

```
    data = r.json()
    assert "keys" in data
    assert len(data["keys"]) >= 1


def test_auth_returns_token():
    r = client.post("/auth")
    assert r.status_code == 200
    assert "token" in r.json()


def test_auth_expired_returns_token():
    r = client.post("/auth?expired=true")
    assert r.status_code == 200
    assert "token" in r.json()


def test_jwks_only_active_keys():
    r = client.get("/jwks")
    keys = r.json()["keys"]
    for k in keys:
        assert "kid" in k
        assert k["kty"] == "RSA"
```

---

✅ **Step 7 — Run Tests + Coverage**

```
pytest --cov=. --cov-report=term-missing
```

Then, starting with Step 2, I use the code as a guide, with a few adjustments that inform me what to write for each file. This manner will also assist me understand what is going on.

In Step 1, I had difficulties importing dependencies because I was using Command Prompt and Bash at the same time, so I asked CHATGPT how to repair it and went through a step-by-step approach.

For example, I copy the problem and paste it onto CHATGPT, to fix it:

“bash: venv/bin/activate: No such file or directory”
**It responded:**

Good — that error tells us exactly what's wrong 👍
You're in **bash**, but your virtual environment was created with the **Windows layout**, not the Linux/Mac layout.

That means the activate script is in:

```
venv/Scripts/activate
```

—not—

```
venv/bin/activate
```

Let's fix this step-by-step.

---

### ✅ Step 1 — Confirm You're in the Right Folder

Run:

```
ls
```

You should see:

```
venv
```

If not, go into your project folder:

```
cd jwks-server
```

---

### ✅ Step 2 — Check What's Inside venv

Run:

```
ls venv
```

You will likely see:

```
Include
Lib
```

```
Scripts
pyvenv.cfg
```

If you see **Scripts**, that confirms it's Windows-style venv.

---

### ✅ Step 3 — Activate Correctly in Bash (Windows + Git Bash)

Use this instead:

```
source venv/Scripts/activate
```

Press Enter.

---

### ✅ Step 4 — Confirm Activation Worked

Your prompt should change to:

```
(venv) user@machine jwks-server $
```

That `(venv)` = success ✅

---

### ❌ If Scripts Folder Doesn't Exist

Then venv wasn't created properly. Recreate it:

```
python -m venv venv
```

Then try again:

```
source venv/Scripts/activate
```

---

### ✅ Step 5 — After Activation Install Packages

```
pip install fastapi uvicorn pyjwt cryptography pytest httpx pytest-cov
```

---

You're actually right on track — this is the most common bash + Windows confusion point.

I also asked this:

"do i have to import this? from cryptography.hazmat.primitives.asymmetric import rsa"

**It responded:**

Yes — ✅ you do need to import that if you are generating RSA key pairs in your JWKS server (which your assignment requires).

Your requirements explicitly say:

> Implement RSA key pair generation
> Serve public keys in JWKS format
> Sign JWTs with RSA keys

So you must generate RSA keys — and that import is how Python's cryptography library provides RSA support.

---

✅ Why This Import Is Needed

```
from cryptography.hazmat.primitives.asymmetric import rsa
```

This gives you access to:

```
rsa.generate_private_key()
```

Which you use like this:

```
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
```

Without this import → your code will crash with:

```
NameError: name 'rsa' is not defined
```

---

✅ Make Sure Package Is Installed

Inside your activated venv, verify:

```
pip show cryptography
```

If not installed:

```
pip install cryptography
```

---

✅ Where It Belongs in Your Project

You should include that import in your keys.py file (or wherever you generate keys).

Example:

```
import time

import uuid

from cryptography.hazmat.primitives.asymmetric import rsa
```

---

✅ Quick Rule for Your Assignment

If your code:

- generates RSA keys → ✅ import rsa

- signs JWT with RS256 → ✅ import rsa
- converts keys to JWKS → ✅ import rsa

So yes — required.