

Kimberley Juarez
11530006
CSCE 3550
February 15, 2026

AI Usage Statement

This is the document containing all the prompts and responses I used to generate this code. I used Copilot to generate everything. The following is the prompts and responses from Copilot.

Prompt 1:

Implementing a basic JWKS Server Objective Develop a RESTful JWKS server that provides public keys with unique identifiers (kid) for verifying JSON Web Tokens (JWTs), implements key expiry for enhanced security, includes an authentication endpoint, and handles the issuance of JWTs with expired keys based on a query parameter. Chooses an appropriate language and web server for the task. Due to the simplicity of this assignment, I would prefer you complete it with an unfamiliar language... but as I have no way to verify it, it's not considered part of the rubric. This project is for educational purposes. In a real-world scenario, you'd want to integrate with a proper authentication system and ensure security best practices. Background HTTP/web servicesLinks to an external site. Familiarize yourself with client/server HTTP services. RESTLinks to an external site. Familiarize yourself with correct HTTP methods/headers/status codes for RESTful APIs. JOSE: JWTLinks to an external site., JWK (and JWKS):Links to an external site. Familiarize yourself with the concepts of JWT, JWK. Understand the importance of key expiry, and kid. Requirements Key Generation Implement RSA key pair generation. Associate a Key ID (kid) and expiry timestamp with each key. Web server with two handlers Serve HTTP on port 8080 A RESTful JWKS endpoint that serves the public keys in JWKS format. Only serve keys that have not expired. A /auth endpoint that returns an unexpired, signed JWT on a POST request. If the "expired" query parameter is present, issue a JWT signed with the expired key pair and the expired expiry. Documentation Code should be organized. Code should be commented where needed. Code should be linted per your language/framework. Tests Test suite for your given language/framework with tests for you. Test coverage should be over 80%. Blackbox testing Ensure the included test clientLinks to an external site. functions against your server. The testing client will attempt a POST to /auth with no body. There is no need to check authentication for this project. NOTE: We are not actually testing user authentication, just mocking authentication and returning a valid JWT for this user Note: Using kid in JWKS is crucial for systems to identify which key to use for JWT verification. Ensure that the JWTs include the kid in their headers and that the JWKS server can serve the correct key when requested with a specific kid. Expected Outcome At the end of the project, you should have a functional JWKS server with a RESTful API that can serve public keys with expiry and unique kid to verify JWTs. The server should authenticate fake users requests, issue JWTs upon successful authentication, and handle the "expired" query parameter to issue JWTs signed

with an expired key. This project should take 1-12 hours, depending on your familiarity with your chosen language/framework, and web servers in general. write everything in python

Response 1:

Here's a complete, minimal-but-solid Python implementation of the JWKS server plus tests.

I'll use:

- **Framework:** FastAPI (clean, typed, easy testing)
- **JWT:** PyJWT
- **RSA keys:** cryptography

You'll get:

- `main.py` — server implementation
- `test_main.py` — tests (pytest)
- `requirements.txt` — dependencies
- Simple in-memory key store with expiry and `kid`
- `/jwks` and `/auth` behavior as specified

`requirements.txt`

```
fastapi
uvicorn
pyjwt
cryptography
pytest
httpx
```

`main.py`

```
from datetime import datetime, timedelta, timezone
from typing import Dict, List, Optional
import uuid

from fastapi import FastAPI, HTTPException, Query
from fastapi.responses import JSONResponse
from pydantic import BaseModel
```

```

import jwt
from jwt import PyJWK
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# -----
# Config
# -----

JWT_ISSUER = "example-issuer"
JWT_AUDIENCE = "example-audience"
JWT_LIFETIME_SECONDS = 300 # 5 minutes
KEY_LIFETIME_SECONDS = 600 # 10 minutes

ALGORITHM = "RS256"

# -----
# Key store and models
# -----

class KeyEntry(BaseModel):
    kid: str
    private_pem: str
    public_pem: str
    expires_at: datetime

class KeyStore:
    """
    In-memory key store with expiry.
    For a real system, you'd persist keys and rotate them properly.
    """

    def __init__(self):
        self._keys: Dict[str, KeyEntry] = {}
        self._expired_key: Optional[KeyEntry] = None

    def _generate_rsa_keypair(self) -> KeyEntry:
        private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
        private_pem = private_key.private_bytes(
            encoding=serialization.Encoding.PEM,

```

```

        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption(),
    ).decode("utf-8")

    public_pem = private_key.public_key().public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo,
    ).decode("utf-8")

    kid = str(uuid.uuid4())
    expires_at = datetime.now(timezone.utc) +
timedelta(seconds=KEY_LIFETIME_SECONDS)

    return KeyEntry(
        kid=kid,
        private_pem=private_pem,
        public_pem=public_pem,
        expires_at=expires_at,
    )

def get_active_key(self) -> KeyEntry:
    """
    Return an unexpired key. If none exists, generate a new one.
    """
    now = datetime.now(timezone.utc)
    # Remove expired keys from active store, but keep the last expired one separately
    expired_kids = []
    for kid, entry in self._keys.items():
        if entry.expires_at <= now:
            expired_kids.append(kid)
            self._expired_key = entry
    for kid in expired_kids:
        del self._keys[kid]

    # If there is at least one active key, return any
    for entry in self._keys.values():
        if entry.expires_at > now:
            return entry

    # Otherwise, generate a new one
    new_entry = self._generate_rsa_keypair()
    self._keys[new_entry.kid] = new_entry
    return new_entry

```

```

def get_expired_key(self) -> Optional[KeyEntry]:
    """
    Return a previously expired key, if any.
    """
    return self._expired_key

def get_unexpired_public_jwks(self) -> Dict:
    """
    Return JWKS containing only unexpired public keys.
    """
    now = datetime.now(timezone.utc)
    keys: List[Dict] = []

    for entry in list(self._keys.values()):
        if entry.expires_at <= now:
            # Move to expired slot
            self._expired_key = entry
            del self._keys[entry.kid]
            continue

        # Convert PEM to JWK
        jwk_obj = PyJWK.from_buffer(entry.public_pem.encode("utf-8"))
        jwk_dict = jwk_obj.to_dict()
        jwk_dict["kid"] = entry.kid
        jwk_dict["alg"] = ALGORITHM
        jwk_dict["use"] = "sig"
        keys.append(jwk_dict)

    return {"keys": keys}

```

```

key_store = KeyStore()
app = FastAPI()

```

```

# -----
# JWT helpers
# -----

```

```

def create_jwt_for_key(entry: KeyEntry, expired: bool = False) -> str:
    """
    Create a JWT signed with the given key.
    If expired=True, the 'exp' claim will be in the past.
    """

```

```

now = datetime.now(timezone.utc)
if expired:
    exp = now - timedelta(seconds=60)
else:
    exp = now + timedelta(seconds=JWT_LIFETIME_SECONDS)

```

```

payload = {
    "sub": "fake-user-id",
    "iss": JWT_ISSUER,
    "aud": JWT_AUDIENCE,
    "iat": int(now.timestamp()),
    "exp": int(exp.timestamp()),
}

```

```

token = jwt.encode(
    payload,
    entry.private_pem,
    algorithm=ALGORITHM,
    headers={"kid": entry.kid},
)
return token

```

```

# -----
# Routes
# -----

```

```

@app.get("/jwks")
def jwks():
    """
    JWKS endpoint.
    Returns only unexpired public keys in JWKS format.
    """
    jwks_body = key_store.get_unexpired_public_jwks()
    return JSONResponse(content=jwks_body)

```

```

@app.post("/auth")
def auth(expired: bool = Query(default=False)):
    """
    Authentication endpoint.
    - Always returns a JWT (no real auth for this assignment).
    - If expired=false (default): sign with an unexpired key and unexpired exp.
    - If expired=true: sign with an expired key and expired exp.
    """

```

```

"""
if not expired:
    key_entry = key_store.get_active_key()
    token = create_jwt_for_key(key_entry, expired=False)
    return {"access_token": token, "token_type": "bearer"}

# expired == True
expired_key = key_store.get_expired_key()
if expired_key is None:
    # Force creation of an active key, then let it expire quickly for demo
    active = key_store.get_active_key()
    # Simulate expiry by setting its expiry in the past
    active.expires_at = datetime.now(timezone.utc) - timedelta(seconds=1)
    # Next call to get_expired_key will pick it up
    expired_key = key_store.get_expired_key()

if expired_key is None:
    raise HTTPException(status_code=500, detail="No expired key available")

token = create_jwt_for_key(expired_key, expired=True)
return {"access_token": token, "token_type": "bearer"}

# -----
# Entry point for running
# -----

# Run with:
# uvicorn main:app --host 0.0.0.0 --port 8080
#
# The assignment specifies port 8080, so use that when starting uvicorn.

```

test_main.py

```

import jwt
from datetime import datetime, timezone
from fastapi.testclient import TestClient

from main import app, ALGORITHM, JWT_ISSUER, JWT_AUDIENCE

client = TestClient(app)

```

```

def test_jwks_returns_valid_structure():
    resp = client.get("/jwks")
    assert resp.status_code == 200
    body = resp.json()
    assert "keys" in body
    assert isinstance(body["keys"], list)

def test_auth_returns_jwt_default_unexpired():
    resp = client.post("/auth")
    assert resp.status_code == 200
    body = resp.json()
    assert "access_token" in body
    token = body["access_token"]

    # Decode without verifying signature, just structure
    header = jwt.get_unverified_header(token)
    assert "kid" in header

    payload = jwt.decode(token, options={"verify_signature": False})
    assert payload["iss"] == JWT_ISSUER
    assert payload["aud"] == JWT_AUDIENCE
    assert payload["sub"] == "fake-user-id"
    assert payload["exp"] > payload["iat"]

def test_auth_expired_query_param():
    # First, get a normal token to ensure a key exists
    resp1 = client.post("/auth")
    assert resp1.status_code == 200

    # Now request an expired token
    resp2 = client.post("/auth?expired=true")
    assert resp2.status_code == 200
    body = resp2.json()
    token = body["access_token"]

    header = jwt.get_unverified_header(token)
    assert "kid" in header

    payload = jwt.decode(token, options={"verify_signature": False})
    assert payload["iss"] == JWT_ISSUER
    assert payload["aud"] == JWT_AUDIENCE
    assert payload["sub"] == "fake-user-id"

```



```

# exp should be in the past
assert payload["exp"] < int(datetime.now(timezone.utc).timestamp())

def test_jwks_contains_unexpired_key_matching_auth_header_kid():
    # Get a fresh token
    resp = client.post("/auth")
    assert resp.status_code == 200
    token = resp.json()["access_token"]
    header = jwt.get_unverified_header(token)
    kid = header["kid"]

    # Fetch JWKS
    jwks_resp = client.get("/jwks")
    assert jwks_resp.status_code == 200
    jwks_body = jwks_resp.json()
    keys = jwks_body["keys"]

    # Ensure at least one key with matching kid
    assert any(k["kid"] == kid for k in keys)

def test_auth_works_with_no_body():
    # The test client in the assignment will POST with no body.
    resp = client.post("/auth")
    assert resp.status_code == 200
    body = resp.json()
    assert "access_token" in body
    assert body["token_type"] == "bearer"

```

How to run

From the project directory:

```

python -m venv venv
source venv/bin/activate # on Windows: venv\Scripts\activate
pip install -r requirements.txt

```

Run the server on port 8080:

```

uvicorn main:app --host 0.0.0.0 --port 8080

```

Run tests (with coverage):

```
pytest --maxfail=1 --disable-warnings -q  
# or with coverage if you install coverage:  
# pip install coverage  
# coverage run -m pytest && coverage report
```

If you want, I can also add a tiny script that acts as a blackbox client similar to your test client to hit `/auth` and `/jwks`.

No further prompts were used, and all code was uploaded to GitHub from there. As stated above, I used Copilot to generate all the code including the test codes. From what I checked, Copilot generated most of what I wanted, but my prompts could use a lot of work. They were minimal and low-effort, so next time I used AI, I will be sure to improve my prompts and get more meticulous about what I ask the AI for.