Client-server testing

Task 1

Question:
Evaluate the code of `app/main.py` and `client.py` and provide a brief overview of potential problems which may affect the reliability and/or performance of the server, client and their interaction. Feel free to provide ideas about fixing said problems.

Answer:

✅ **Summary**

- A FastAPI service (**main.py**) that proxies metadata from another service (http://data/client-libraries/).
- A Python client (**client.py)** that fetches that metadata, parses it, and checks Python compatibility on PyPI.

☐ **Key issues to fix as improvements:**

☐ **main.py (FastAPI server)**

☐ **Old:**

```
from urllib.request import urlopen
import json
@app.get("/")
def iqm_client_metadata():
    url = "http://data/client-libraries/"
    data = json.loads(urlopen(url).read().decode("utf-8"))
    return {"client": data["iqm_client"]}
```

☐ **Problems:**

- **Blocking I/O: urlopen()** is a synchronous, blocking call — not ideal in a FastAPI app, which is built for **asynchronous concurrency**,which is async-native.
- **Authentication:** Add HTTP Basic Auth, JWT, OAuth2 or token-based auth to the **main.py** route if needed.
- **No timeouts:** Could hang indefinitely on network delays.
- **No error handling:** Any failure would crash the API with a 500.
- **Client_key parameter not added.**
- **Hardcoded client key:** Always returns "**iqm_client**" only.

✅**New:**

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

```
async def iqm_client_metadata(client_key: Optional[str] = None):
    ...
```

✅**Improvements:**

- **Async compatible**: Ready for FastAPI's async event loop.
- **Optional[str] param:** Lets the user specify a different client_key.
- **Safe networking**: **Timeout** and **httpx** library or **requests library** with **raise_for_status()** guard against issues instead of **urlopen()** or **urllib.request a**nd makes error handling explicit and maintainable.
- Add logging, retry-logic and environmental variables like **os.getenv().**
- **HTTPException:** FastAPI error responses (with status codes and messages)

client.py (Python client for interacting with a FastAPI backend.)

Old:

```
import urllib.request
import base64
import json

def main():
    request = urllib.request.Request("http://localhost:8000/")
    ...
    data = json.loads(urllib.request.urlopen(request).read().decode("utf-8"))
```

Problems:

- Not used **httpx** Instead of **urllib.request.**
- Not used **requests** or **httpx** With Built-in Basic Auth.
- Handle Exceptions missing.
- No separate Functions for Reusability and Testing.
- **Timeout & Retries:** No added timeouts and retries.

✅**New:**

```
import httpx

def get_client_data(...)
def get_python_version_from_pypi(...)
def main():
    try:
        ...
```

✅Improvements:

- **Modern HTTP client**: Uses httpx with raise_for_status().
- Use **httpx** Instead of **urllib.request**
- **Timeouts & Retries:** Prevents hangs on slow responses.
- **Explicit exception handling**: Catches and prints specific error types.
- **Reusable components**: **get_client_data()** and **get_python_version_from_pypi()** can be reused or tested independently. Separate functions for re-usability and testing

☐ Final Thoughts as summary table

Area	**Original Version**	**Improved Version**	**Benefits**
HTTP Client	urllib.request.urlopen()	httpx.get()	Modern, feature-rich, easier error handling, async-ready
Error Handling	None or bare except	Structured with specific exceptions	Prevents crashes, improves debuggability
Timeouts	Not set (defaults to forever)	Explicit timeout=10.0	Prevents hanging requests
Authorization	Basic auth manually encoded	Still manual, but structured	Still good, but consider httpx.BasicAuth in future
Async I/O	Blocking I/O in FastAPI	Async using httpx.AsyncClient optional	Now ready for scaling and non-blocking behavior
Optional Parameters	Not supported	client_key query param added for ex: GET /?client_key=iqm_client (main.py)	Flexible API usage
Response validation	None	Now checks for key existence For ex: {"error": "unauthorized"} # or missing `iqm_client`	Prevents KeyError crashes
Centralized error handling	No	Yes – handles request, status, and unexpected errors	Clean control flow, better for production logs