Enterprise SDK & API Documentation for Post-Quantum Cryptography with Python Concurrency and Developer Enablement

Overview:

This repository demonstrates **enterprise-grade SDK and REST API documentation** for a **Post-Quantum Cryptography (PQC) key exchange system based on the Kyber1024 algorithm**, fully compliant with **NIST PQC standards**. The sample is designed for **developers, security engineers, and internal technical teams** requiring **secure, future-proof, and high-performance cryptographic integrations**.

This documentation is written in **Markdown with OpenAPI 3.1 integration**, reflecting **Meta Reality Labs-level documentation standards**. This includes **developer enablement, agent readiness workflows, and API adoption guidance on a large scale,** which is all conveyed through the use of the advanced technical writing, high-impact SEO, and practical code samples.

Key Features & Highlights

- **SDK/API Documentation Excellence**
    - Full **OpenAPI 3.1 specification** with endpoints, methods, parameters, and example payloads.
    - **RESTful endpoints** supporting Kyber1024 key generation, encryption, and decryption.
    - **OAuth2 authentication** with bearer token examples and security best practices.
    - Response codes, error handling, rate limiting, and versioning fully documented.
- **Advanced Technical Concepts**
    - **Post-Quantum Cryptography** (Kyber1024) for quantum-resistant key exchange.

    - **Python concurrency patterns** for secure key management, asynchronous key generation, and batch processing.
    - Integration with **multi-tenant cloud systems** for scalable, enterprise-ready deployments.

- ## Developer Enablement
    - Step-by-step **SDK onboarding guide**, including setup, environment configuration, and usage examples.
    - **Python sample scripts** demonstrating key exchange, encryption/decryption, and error handling.
    - Multi-platform **integration tips** for web, mobile, and cloud applications.

- ## Enterprise Documentation Architecture
    - **Markdown + OpenAPI 3.1 + YAML workflows**, structured for **internal knowledge bases, Confluence-ready outputs, and public Help Centers**.
    - Diagrammed workflows using **Mermaid and PlantUML** for secure key exchange pipelines.
    - **Changelog and versioning standards** for release management and content lifecycle tracking.

- ## SEO & Discoverability Optimized
    - High-ranking keywords embedded: SDK documentation, OpenAPI 3.1, post-quantum cryptography, developer enablement, Python concurrency, Kyber1024, quantum-resistant key exchange, cloud infrastructure documentation.
    - Internal linking for cross-referenced endpoints and SDK examples.
    - Structured data and modular headings for enhanced search discoverability.

Sample Endpoint Documentation (Excerpt)

Endpoint: /api/v1/keys/generate
Method: POST
Description: Generates a quantum-resistant public/private key pair using Kyber1024.

Request Example:

POST /api/v1/keys/generate

Authorization: Bearer <token>

Content-Type: application/json

```
{
  "client_id": "dev_team_01",
  "key_type": "kyber1024",
  "expiry_days": 365
}
```

Response Example:

```
{
  "key_id": "abcd1234",
  "public_key": "<base64_public_key>",
  "private_key": "<base64_private_key>",
  "created_at": "2025-08-28T14:30:00Z",
  "expires_at": "2026-08-28T14:30:00Z"
}
```

Error Codes:

- 401 Unauthorized – Invalid bearer token.
- 429 Too Many Requests – Rate limit exceeded.
- 500 Internal Server Error – Key generation failed.

Python SDK Sample (Concurrent Key Generation)

```
import asyncio
```

```
from pqc_sdk import KyberKeyManager

async def generate_keys(client_id: str, n_keys: int):
    manager = KyberKeyManager(client_id)
    tasks = [manager.generate_key_async() for _ in range(n_keys)]
    results = await asyncio.gather(*tasks)
    for key in results:
        print(f"Key ID: {key.key_id}, Public Key: {key.public_key[:10]}...")

# Run concurrent generation for 10 keys
asyncio.run(generate_keys("dev_team_01", 10))
```