Tính năng và Phương pháp kiểm tra cho từng Microservice

Sau khi phân tích cấu trúc thư mục chi tiết cho từng Microservice, chúng tôi sẽ đề xuất các tính năng cốt lõi và phương pháp kiểm tra tương ứng để đảm bảo chất lượng và hoạt động ổn định của hệ thống sau khi triển khai.

1. User Service

Mục đích: Quản lý tất cả các nghiệp vụ liên quan đến người dùng: đăng ký, đăng nhập, quản lý thông tin cá nhân (hồ sơ), quản lý vai trò (user, admin), và quản lý số dư tài khoản.

1.1. Tính năng

- **Đăng ký tài khoản:** Cho phép người dùng mới tạo tài khoản với các thông tin cơ bản như email, mật khẩu, tên. Hệ thống cần xác thực email và đảm bảo mật khẩu đủ mạnh.
- **Đăng nhập tài khoản:** Cung cấp cơ chế xác thực người dùng bằng email và mật khẩu, trả về JWT (JSON Web Token) để các dịch vụ khác có thể xác thực phiên làm việc.
- **Quản lý hồ sơ người dùng:** Cho phép người dùng xem và cập nhật thông tin cá nhân (tên, địa chỉ, số điện thoại, v.v.).
- **Quản lý vai trò:** Cung cấp chức năng cho quản trị viên để gán hoặc thay đổi vai trò của người dùng (ví dụ: User, Admin).
- **Quản lý số dư tài khoản:** Theo dõi và cập nhật số dư tài khoản của người dùng, tích hợp với Payment Service để ghi nhận các giao dịch nạp/rút tiền.
- **Quên mật khẩu/Đặt lại mật khẩu:** Cung cấp quy trình an toàn để người dùng có thể đặt lại mật khẩu khi quên, thường thông qua email xác thực.
- **Xác thực hai yếu tố (2FA):** Tùy chọn tăng cường bảo mật bằng cách yêu cầu mã xác thực bổ sung ngoài mật khẩu.

1.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các hàm xử lý logic nghiệp vụ riêng lẻ (ví dụ: hàm mã hóa mật khẩu, hàm tạo JWT, hàm xác thực email). **Công cụ:** `pytest` (https://pytest.org/) (Python), `unittest` (https://docs.python.org/3/library/unittest.html) (Python).
  - Kiểm tra các hàm CRUD (Create, Read, Update, Delete) của `user.py` để đảm bảo tương tác đúng với database. **Công cụ:** `pytest` (https://pytest.org/) với `pytest-mock` (https://github.com/pytest-dev/pytest-mock) hoặc `unittest.mock`

(https://docs.python.org/3/library/unittest.mock.html) để mock database connection, hoặc sử dụng database in-memory như SQLite cho các test đơn giản.

- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra luồng đăng ký/đăng nhập hoàn chỉnh, bao gồm việc gọi API, xác thực dữ liệu, và lưu trữ vào database. **Công cụ:** `pytest` (https://pytest.org/) với `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/) để gọi API, `SQLAlchemy` (https://www.sqlalchemy.org/) để tương tác với database thật hoặc test database.
  - Kiểm tra luồng cập nhật hồ sơ người dùng, đảm bảo dữ liệu được cập nhật chính xác và các ràng buộc được tuân thủ. **Công cụ:** Tương tự như trên.
  - Kiểm tra tích hợp với Payment Service khi số dư tài khoản được cập nhật sau giao dịch. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) để gọi API của Payment Service, có thể sử dụng `pytest-respx` (https://lundberg.github.io/respx/) để mock các phản hồi từ Payment Service nếu không muốn gọi dịch vụ thật.
  - Kiểm tra luồng quên mật khẩu, đảm bảo email xác thực được gửi và mật khẩu có thể đặt lại thành công. **Công cụ:** `pytest` (https://pytest.org/), mock dịch vụ gửi email (ví dụ: `smtplib` hoặc thư viện email).
- **Kiểm thử API (API Testing):**
  - Sử dụng các công cụ như Postman (https://www.postman.com/downloads/), cURL (https://curl.se/), hoặc thư viện `requests` (https://requests.readthedocs.io/) trong Python để gửi các yêu cầu HTTP đến các endpoint của User Service (ví dụ: `/api/v1/auth/register`, `/api/v1/auth/login`, `/api/v1/users/me`). **Công cụ:** Postman (https://www.postman.com/downloads/), Insomnia (https://insomnia.rest/download), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python), **Pytest-API** (https://github.com/NAVEENINTEL/python-pytest-api-testing), **Robot Framework** (https://robotframework.org/) (với thư viện RequestsLibrary).
  - Kiểm tra các mã trạng thái HTTP (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found). **Cách kiểm tra:** Phân tích phản hồi HTTP status code.
  - Kiểm tra cấu trúc và nội dung của phản hồi JSON. **Cách kiểm tra:** Sử dụng JSON schema validation (ví dụ: thư viện `jsonschema` trong Python) hoặc kiểm tra từng trường cụ thể.
  - Kiểm tra các trường hợp biên (edge cases) như dữ liệu đầu vào không hợp lệ, thiếu trường bắt buộc, hoặc giá trị vượt quá giới hạn. **Cách kiểm tra:** Tạo các test case với dữ liệu đầu vào không hợp lệ và mong đợi các lỗi phù hợp.

- **Kiểm thử bảo mật (Security Testing):**
  - Kiểm tra các lỗ hổng phổ biến như SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, Insecure Direct Object References (IDOR). **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite (https://portswigger.net/burp/communitydownload), Nessus (https://www.tenable.com/downloads/nessus), SonarQube (https://www.sonarsource.com/products/sonarqube/downloads/) (phân tích mã tĩnh).
  - Đảm bảo JWT được tạo và xác thực đúng cách, không dễ bị giả mạo hoặc hết hạn. **Cách kiểm tra:** Thử sửa đổi JWT, sử dụng JWT hết hạn, hoặc JWT không hợp lệ để truy cập tài nguyên.
  - Kiểm tra chính sách mật khẩu mạnh và cơ chế lưu trữ mật khẩu an toàn (sử dụng bcrypt). **Cách kiểm tra:** Thử đăng ký với mật khẩu yếu, kiểm tra xem mật khẩu có được hash đúng cách trong database.
  - Kiểm tra phân quyền (Authorization) để đảm bảo chỉ người dùng có vai trò phù hợp mới có thể truy cập các tài nguyên hoặc thực hiện các hành động nhất định (ví dụ: chỉ Admin mới có thể quản lý vai trò). **Cách kiểm tra:** Thử truy cập các API yêu cầu quyền Admin bằng tài khoản User thông thường.
- **Kiểm thử hiệu năng (Performance Testing):**
  - Sử dụng các công cụ như JMeter, Locust (https://locust.io/) để mô phỏng tải người dùng đồng thời lên các API quan trọng (đăng ký, đăng nhập, cập nhật hồ sơ). **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
  - Đo lường thời gian phản hồi, thông lượng (throughput), và tỷ lệ lỗi dưới các mức tải khác nhau. **Cách kiểm tra:** Chạy các kịch bản tải với số lượng người dùng đồng thời tăng dần và theo dõi các chỉ số hiệu năng.
  - Xác định các điểm nghẽn (bottlenecks) và đánh giá khả năng mở rộng của dịch vụ. **Cách kiểm tra:** Phân tích kết quả kiểm thử hiệu năng, sử dụng các công cụ giám sát (Prometheus, Grafana) để theo dõi tài nguyên hệ thống.
- **Kiểm thử khả năng phục hồi (Resilience Testing):**
  - Kiểm tra hành vi của dịch vụ khi database không khả dụng hoặc có độ trễ cao. **Công cụ:** Chaos Monkey (https://github.com/Netflix/chaosmonkey) (Netflix), Gremlin (https://www.gremlin.com/), hoặc mô phỏng lỗi thủ công (ví dụ: tắt database server).
  - Kiểm tra khả năng xử lý lỗi và trả về thông báo lỗi thân thiện cho người dùng. **Cách kiểm tra:** Gây ra lỗi có chủ đích và kiểm tra phản hồi của API.

2. Payment Service

Mục đích: Xử lý tất cả các nghiệp vụ liên quan đến nạp tiền, thanh toán, quản lý giao dịch, và tích hợp với các cổng thanh toán bên ngoài.

2.1. Tính năng

- **Nạp tiền vào tài khoản:** Cho phép người dùng nạp tiền vào số dư tài khoản của họ thông qua các cổng thanh toán tích hợp (VNPay, Momo, Bank API).
- **Thanh toán dịch vụ:** Cho phép người dùng sử dụng số dư tài khoản để thanh toán cho các dịch vụ toán học hoặc các gói dịch vụ khác.
- **Quản lý giao dịch:** Lưu trữ và cung cấp lịch sử chi tiết các giao dịch nạp tiền và thanh toán của người dùng.
- **Tích hợp cổng thanh toán:** Kết nối và xử lý giao tiếp với các cổng thanh toán bên ngoài (VNPay, Momo, API ngân hàng) để thực hiện các giao dịch.
- **Xử lý Webhook:** Nhận và xử lý các thông báo (webhook) từ cổng thanh toán về trạng thái giao dịch (thành công, thất bại, đang chờ xử lý).
- **Cập nhật số dư người dùng:** Đảm bảo số dư tài khoản của người dùng được cập nhật chính xác và kịp thời sau mỗi giao dịch thành công hoặc thất bại.
- **Hoàn tiền (Refund):** Cung cấp chức năng hoàn tiền cho các giao dịch đã thực hiện (nếu có yêu cầu và chính sách cho phép).

2.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các hàm xử lý logic nghiệp vụ liên quan đến tính toán số dư, xác thực giao dịch. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm CRUD cho `transaction.py` và `payment_package.py`. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm xử lý phản hồi từ cổng thanh toán (parsing webhook data). **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra toàn bộ luồng nạp tiền từ khi người dùng khởi tạo yêu cầu đến khi số dư được cập nhật, bao gồm cả việc tương tác với cổng thanh toán giả lập (mock payment gateway). **Công cụ:** `pytest` (https://pytest.org/), `httpx`

(https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/), `pytest-respx` (https://lundberg.github.io/respx/) (để mock cổng thanh toán).
    - Kiểm tra luồng thanh toán dịch vụ, đảm bảo số dư bị trừ chính xác và giao dịch được ghi nhận. **Công cụ:** Tương tự như trên.
    - Kiểm tra tích hợp với User Service để cập nhật số dư người dùng. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/).
    - Kiểm tra xử lý webhook, đảm bảo dịch vụ nhận và xử lý đúng các thông báo từ cổng thanh toán. **Cách kiểm tra:** Gửi các yêu cầu webhook giả lập đến dịch vụ và kiểm tra phản hồi, trạng thái giao dịch.
- **Kiểm thử API (API Testing):**
    - Kiểm tra các endpoint liên quan đến nạp tiền, thanh toán, truy vấn lịch sử giao dịch. **Công cụ:** Postman (https://www.postman.com/downloads/), Insomnia (https://insomnia.rest/download), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python), **SoapUI** (https://www.soapui.org/downloads/soapui/) (nếu có SOAP API), **Karate DSL** (https://www.karatelabs.io/).
    - Kiểm tra các trường hợp lỗi như giao dịch thất bại, số dư không đủ, thông tin thanh toán không hợp lệ. **Cách kiểm tra:** Gửi các yêu cầu với dữ liệu không hợp lệ hoặc mô phỏng trạng thái lỗi từ cổng thanh toán.
    - Kiểm tra bảo mật của các API webhook để đảm bảo chỉ các yêu cầu hợp lệ từ cổng thanh toán mới được xử lý. **Cách kiểm tra:** Thử gửi webhook từ các nguồn không xác định hoặc với chữ ký không hợp lệ.
- **Kiểm thử bảo mật (Security Testing):**
    - Đảm bảo các thông tin nhạy cảm về thanh toán được mã hóa và bảo vệ đúng cách. **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite (https://portswigger.net/burp/communitydownload).
    - Kiểm tra các lỗ hổng liên quan đến giao dịch như giả mạo yêu cầu, tấn công replay. **Cách kiểm tra:** Thử gửi lại các yêu cầu giao dịch cũ, thay đổi thông tin giao dịch.
    - Đảm bảo các cơ chế xác thực và ủy quyền được áp dụng chặt chẽ cho các API thanh toán. **Cách kiểm tra:** Thử truy cập các API thanh toán mà không có quyền hoặc với quyền không đủ.
- **Kiểm thử hiệu năng (Performance Testing):**
    - Đánh giá khả năng xử lý đồng thời các giao dịch nạp/thanh toán dưới tải cao. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
    - Đo lường thời gian phản hồi của các API giao dịch và API webhook. **Cách kiểm tra:** Chạy các kịch bản tải và theo dõi thời gian phản hồi.

- **Kiểm thử khả năng phục hồi (Resilience Testing):**
  - Kiểm tra hành vi của dịch vụ khi cổng thanh toán bên ngoài không phản hồi hoặc gặp lỗi. **Công cụ:** Chaos Monkey (https://github.com/Netflix/chaosmonkey), Gremlin (https://www.gremlin.com/), hoặc mô phỏng lỗi thủ công.
  - Đảm bảo cơ chế retry (thử lại) và xử lý lỗi được triển khai để tránh mất mát dữ liệu hoặc giao dịch bị kẹt. **Cách kiểm tra:** Gây lỗi cổng thanh toán và kiểm tra xem dịch vụ có thử lại hoặc ghi nhận lỗi đúng cách không.

3. Math Solver Service

Mục đích: Chứa logic giải các bài toán toán học và cung cấp API để Frontend hoặc các ứng dụng khác có thể gọi đến.

3.1. Tính năng

- **Giải phương trình bậc hai:** Cung cấp API để nhận các tham số của phương trình bậc hai (a, b, c) và trả về nghiệm của phương trình.
- **Giải hệ phương trình tuyến tính:** Cung cấp API để nhận các tham số của hệ phương trình tuyến tính và trả về nghiệm của hệ.
- **Lưu trữ lịch sử giải toán:** Ghi lại các bài toán đã được giải và kết quả tương ứng, có thể liên kết với người dùng đã thực hiện yêu cầu.
- **Hỗ trợ nhiều loại bài toán:** Mở rộng để giải các loại bài toán khác như đạo hàm, tích phân, ma trận, v.v.
- **Xử lý lỗi đầu vào:** Đảm bảo dịch vụ có thể xử lý các đầu vào không hợp lệ (ví dụ: tham số không phải số, phương trình vô nghiệm/vô số nghiệm) và trả về thông báo lỗi rõ ràng.

3.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các hàm giải phương trình bậc hai với các trường hợp: có 2 nghiệm phân biệt, có nghiệm kép, vô nghiệm. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm giải hệ phương trình tuyến tính với các trường hợp: có nghiệm duy nhất, vô nghiệm, vô số nghiệm. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm xử lý đầu vào không hợp lệ (ví dụ: a=0 cho phương trình bậc hai). **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).

    - Kiểm tra các hàm CRUD của `math_solution.py`. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
- **Kiểm thử tích hợp (Integration Testing):**
    - Kiểm tra luồng từ khi nhận yêu cầu giải toán đến khi lưu trữ kết quả vào database. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/).
    - Kiểm tra tích hợp với User Service để liên kết lịch sử giải toán với người dùng. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/).
- **Kiểm thử API (API Testing):**
    - Kiểm tra các endpoint giải phương trình bậc hai (`/api/v1/math/quadratic`) và hệ phương trình tuyến tính (`/api/v1/math/linear_system`). **Công cụ:** Postman (https://www.postman.com/downloads/), Insomnia (https://insomnia.rest/download), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python), **ReadyAPI** (https://support.smartbear.com/readyapi/downloads/), **JMeter** (https://jmeter.apache.org/) (cho API).
    - Kiểm tra các phản hồi JSON cho các trường hợp thành công và thất bại. **Cách kiểm tra:** Phân tích phản hồi HTTP status code và nội dung JSON.
    - Kiểm tra các trường hợp biên về giá trị đầu vào (số rất lớn, số rất nhỏ, số âm, số 0). **Cách kiểm tra:** Gửi các yêu cầu với các giá trị đầu vào biên và kiểm tra kết quả.
- **Kiểm thử hiệu năng (Performance Testing):**
    - Đánh giá thời gian phản hồi của các API giải toán dưới tải cao, đặc biệt với các bài toán phức tạp. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
    - Kiểm tra khả năng xử lý đồng thời nhiều yêu cầu giải toán. **Cách kiểm tra:** Chạy các kịch bản tải với số lượng người dùng đồng thời tăng dần.
- **Kiểm thử độ chính xác (Accuracy Testing):**
    - Sử dụng bộ dữ liệu kiểm thử lớn với các bài toán có nghiệm đã biết để xác minh độ chính xác của các thuật toán giải toán. **Cách kiểm tra:** So sánh kết quả trả về từ dịch vụ với kết quả mong đợi từ các nguồn đáng tin cậy hoặc tính toán thủ công.
    - So sánh kết quả của dịch vụ với kết quả tính toán thủ công hoặc từ các công cụ toán học đáng tin cậy. **Công cụ:** Python với thư viện `numpy`, `sympy` để tính toán và so sánh.

4. Content Service

Mục đích: Quản lý tất cả nội dung tĩnh của website như trang chủ, giới thiệu, liên hệ, FAQ, chính sách bảo mật, điều khoản sử dụng.

4.1. Tính năng

- **Quản lý trang tĩnh:** Cho phép tạo, đọc, cập nhật, xóa (CRUD) các trang nội dung tĩnh như "Giới thiệu", "Liên hệ", "Chính sách bảo mật".
- **Quản lý FAQ:** Cho phép tạo, đọc, cập nhật, xóa (CRUD) các câu hỏi thường gặp và câu trả lời tương ứng.
- **Phân loại nội dung:** Hỗ trợ phân loại nội dung theo danh mục hoặc thẻ để dễ dàng quản lý và tìm kiếm.
- **Hỗ trợ đa ngôn ngữ:** Tùy chọn cung cấp nội dung bằng nhiều ngôn ngữ khác nhau.
- **Tích hợp trình soạn thảo:** Cho phép quản trị viên sử dụng trình soạn thảo WYSIWYG để tạo và chỉnh sửa nội dung.

4.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các hàm CRUD cho `page.py` và `faq.py`. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm xử lý phân loại nội dung. **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra luồng tạo/cập nhật một trang tĩnh hoặc một FAQ, đảm bảo dữ liệu được lưu trữ và truy xuất chính xác. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/).
  - Kiểm tra tích hợp với Admin Service để quản trị viên có thể quản lý nội dung. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/).
- **Kiểm thử API (API Testing):**
  - Kiểm tra các endpoint để lấy danh sách trang, chi tiết trang, danh sách FAQ, chi tiết FAQ. **Công cụ:** Postman (https://www.postman.com/downloads/), Insomnia (https://insomnia.rest/download), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python), **Newman** (https://learning.postman.com/docs/collections/using-newman-cli/command-line-integration-with-newman/) (cho Postman (https://www.postman.com/downloads/) Collection Runner).
  - Kiểm tra các endpoint để tạo, cập nhật, xóa trang/FAQ (yêu cầu xác thực và phân

quyền). **Cách kiểm tra:** Gửi các yêu cầu với token xác thực và kiểm tra phản hồi.
    - Kiểm tra các trường hợp lỗi như ID không tồn tại, dữ liệu không hợp lệ. **Cách kiểm tra:** Gửi các yêu cầu với dữ liệu không hợp lệ và kiểm tra lỗi trả về.
- **Kiểm thử bảo mật (Security Testing):**
    - Đảm bảo chỉ quản trị viên có quyền mới có thể tạo, cập nhật, xóa nội dung. **Cách kiểm tra:** Thử truy cập các API quản lý nội dung bằng tài khoản không có quyền Admin.
    - Kiểm tra các lỗ hổng XSS khi hiển thị nội dung được tạo bởi người dùng (nếu có). **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite (https://portswigger.net/burp/communitydownload). **Cách kiểm tra:** Chèn các script độc hại vào nội dung và kiểm tra xem chúng có được thực thi khi hiển thị không.
- **Kiểm thử hiệu năng (Performance Testing):
    - Đánh giá thời gian phản hồi khi truy xuất các trang nội dung tĩnh hoặc danh sách FAQ dưới tải cao. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
- **Kiểm thử khả năng phục hồi (Resilience Testing):**
    - Kiểm tra hành vi của dịch vụ khi database không khả dụng. **Công cụ:** Chaos Monkey (https://github.com/Netflix/chaosmonkey), Gremlin (https://www.gremlin.com/), hoặc mô phỏng lỗi thủ công. **Cách kiểm tra:** Tắt database và kiểm tra phản hồi của dịch vụ.

5. Admin Service

Mục đích: Cung cấp các chức năng quản trị hệ thống: quản lý người dùng, quản lý dịch vụ, quản lý giao dịch, thống kê báo cáo, quản lý quyền truy cập. Đây là dịch vụ tổng hợp, gọi đến các microservice khác để lấy dữ liệu và thực hiện các thao tác quản trị.

5.1. Tính năng

- **Quản lý người dùng:** Cho phép quản trị viên xem, tìm kiếm, chỉnh sửa thông tin người dùng, khóa/mở khóa tài khoản, thay đổi vai trò người dùng (thông qua User Service).
- **Quản lý dịch vụ:** Cho phép quản trị viên quản lý các dịch vụ toán học hoặc các gói dịch vụ khác (ví dụ: thêm, sửa, xóa dịch vụ).
- **Quản lý giao dịch:** Xem lịch sử giao dịch của tất cả người dùng, tìm kiếm giao dịch theo tiêu chí, xem chi tiết giao dịch (thông qua Payment Service).

- **Thống kê báo cáo:** Cung cấp các báo cáo tổng hợp về số lượng người dùng, doanh thu, số lượng giao dịch, hiệu suất dịch vụ.
- **Quản lý quyền truy cập:** Quản lý các tài khoản quản trị viên, phân quyền cho từng quản trị viên dựa trên vai trò.
- **Quản lý nội dung:** Cho phép quản trị viên tạo, chỉnh sửa, xóa các nội dung tĩnh của website (thông qua Content Service).
- **Ghi log hoạt động:** Ghi lại các hoạt động quan trọng của quản trị viên để phục vụ mục đích kiểm toán và bảo mật.

5.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các hàm xử lý logic nghiệp vụ riêng của Admin Service (ví dụ: hàm tổng hợp báo cáo, hàm phân quyền). **Công cụ:** `pytest` (https://pytest.org/), `unittest` (https://docs.python.org/3/library/unittest.html).
  - Kiểm tra các hàm gọi API đến các microservice khác (User Service, Payment Service, Content Service). **Công cụ:** `pytest` (https://pytest.org/) với `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/), có thể sử dụng `pytest-respx` (https://lundberg.github.io/respx/) để mock các phản hồi từ các dịch vụ khác.
- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra toàn bộ luồng quản lý người dùng, bao gồm việc gọi API đến User Service và hiển thị dữ liệu trên giao diện quản trị. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/).
  - Kiểm tra luồng quản lý giao dịch, đảm bảo dữ liệu từ Payment Service được hiển thị chính xác và các thao tác quản lý được thực hiện đúng. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/).
  - Kiểm tra luồng tạo/chỉnh sửa nội dung thông qua Content Service. **Công cụ:** `pytest` (https://pytest.org/), `httpx` (https://www.python-httpx.org/) hoặc `requests` (https://requests.readthedocs.io/).
  - Kiểm tra các báo cáo thống kê, đảm bảo dữ liệu được tổng hợp chính xác từ các microservice liên quan. **Cách kiểm tra:** So sánh dữ liệu báo cáo với dữ liệu gốc từ các microservice.
- **Kiểm thử API (API Testing):**

    - Kiểm tra các endpoint của Admin Service, đảm bảo các yêu cầu được chuyển tiếp đúng đến các microservice backend và phản hồi được xử lý chính xác. **Công cụ:** Postman (https://www.postman.com/downloads/), Insomnia (https://insomnia.rest/download), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python), **Swagger UI** (https://swagger.io/tools/swagger-ui/) (để khám phá API), **Dredd** (https://github.com/apiaryio/dredd) (cho API contract testing).
    - Kiểm tra các trường hợp lỗi khi một microservice backend không khả dụng hoặc trả về lỗi. **Cách kiểm tra:** Mô phỏng lỗi từ các microservice backend (ví dụ: tắt dịch vụ, trả về lỗi 500) và kiểm tra phản hồi của Admin Service.
    - Kiểm tra phân quyền truy cập API, đảm bảo chỉ quản trị viên có quyền mới có thể thực hiện các thao tác quản trị. **Cách kiểm tra:** Thử truy cập các API quản trị bằng tài khoản không có quyền hoặc quyền không đủ.
- **Kiểm thử bảo mật (Security Testing):**
    - Đảm bảo cơ chế xác thực và ủy quyền cho tài khoản quản trị viên là mạnh mẽ. **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite (https://portswigger.net/burp/communitydownload).
    - Kiểm tra các lỗ hổng liên quan đến truy cập trái phép vào các chức năng quản trị. **Cách kiểm tra:** Thử các kỹ thuật tấn công như brute-force, session hijacking.
    - Kiểm tra việc ghi log hoạt động của quản trị viên, đảm bảo tính toàn vẹn và không thể bị giả mạo. **Cách kiểm tra:** Thử sửa đổi log hoặc thực hiện hành động mà không được ghi log.
- **Kiểm thử hiệu năng (Performance Testing):**
    - Đánh giá thời gian phản hồi của các API quản trị, đặc biệt là các API tổng hợp dữ liệu từ nhiều microservice. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
    - Kiểm tra khả năng xử lý đồng thời nhiều yêu cầu từ các quản trị viên. **Cách kiểm tra:** Chạy các kịch bản tải với nhiều quản trị viên đồng thời.
- **Kiểm thử khả năng phục hồi (Resilience Testing):**
    - Kiểm tra hành vi của Admin Service khi một hoặc nhiều microservice backend không khả dụng. **Công cụ:** Chaos Monkey (https://github.com/Netflix/chaosmonkey), Gremlin (https://www.gremlin.com/), hoặc mô phỏng lỗi thủ công.
    - Đảm bảo các cơ chế xử lý lỗi và thông báo được triển khai để quản trị viên có thể nhận biết và xử lý sự cố. **Cách kiểm tra:** Gây lỗi cho các microservice phụ thuộc và kiểm tra thông báo lỗi của Admin Service.

6. Frontend (Web App)

Mục đích: Xây dựng giao diện người dùng tương tác trực tiếp với người dùng cuối, thông qua API Gateway để giao tiếp với các Microservice Backend.

6.1. Tính năng

- **Giao diện người dùng thân thiện:** Cung cấp giao diện trực quan, dễ sử dụng cho người dùng cuối để tương tác với các dịch vụ toán học, quản lý tài khoản, và thực hiện thanh toán.
- **Đăng ký/Đăng nhập/Đăng xuất:** Cho phép người dùng thực hiện các thao tác xác thực tài khoản một cách dễ dàng và an toàn.
- **Quản lý hồ sơ cá nhân:** Hiển thị và cho phép người dùng cập nhật thông tin cá nhân của họ.
- **Lịch sử giao dịch/sử dụng dịch vụ:** Hiển thị lịch sử các giao dịch nạp tiền, thanh toán và lịch sử sử dụng các dịch vụ giải toán.
- **Giao diện giải toán:** Cung cấp các form nhập liệu và hiển thị kết quả cho các bài toán toán học (phương trình bậc hai, hệ phương trình tuyến tính).
- **Tích hợp API:** Giao tiếp với các Microservice Backend thông qua API Gateway để lấy dữ liệu và gửi yêu cầu.
- **Quản lý trạng thái (State Management):** Quản lý trạng thái ứng dụng (ví dụ: trạng thái xác thực người dùng, dữ liệu người dùng, trạng thái giỏ hàng) một cách hiệu quả.
- **Hiển thị nội dung tĩnh:** Hiển thị các trang nội dung tĩnh như trang chủ, giới thiệu, FAQ, chính sách bảo mật.
- **Phản hồi người dùng:** Cung cấp phản hồi trực quan cho người dùng về trạng thái của các thao tác (ví dụ: thông báo thành công, lỗi, đang tải).
- **Thiết kế đáp ứng (Responsive Design):** Đảm bảo giao diện hiển thị tốt trên nhiều thiết bị và kích thước màn hình khác nhau (desktop, tablet, mobile).

6.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các component UI riêng lẻ (ví dụ: nút, input, modal) để đảm bảo chúng hoạt động đúng và hiển thị chính xác. **Công cụ:** Jest, React Testing Library (cho React), Vue Test Utils (cho Vue), Jasmine, Mocha, Karma (cho JavaScript nói chung).
  - Kiểm tra các custom React Hooks và các hàm tiện ích (utils) để đảm bảo logic hoạt động đúng. **Công cụ:** Jest, React Testing Library.

- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra luồng tương tác giữa các component (ví dụ: form đăng nhập gửi dữ liệu đến API và hiển thị thông báo). **Công cụ:** Jest, React Testing Library, Cypress (cho các kịch bản phức tạp hơn).
  - Kiểm tra tích hợp với các API backend (sử dụng mock API hoặc môi trường staging) để đảm bảo dữ liệu được gửi và nhận đúng định dạng. **Công cụ:** Mock Service Worker (MSW), Nock (Node.js), hoặc sử dụng môi trường staging/test.
  - Kiểm tra luồng người dùng hoàn chỉnh (end-to-end flows) như đăng ký, đăng nhập, nạp tiền, giải toán, xem lịch sử. **Công cụ:** Cypress, Playwright, Selenium.
- **Kiểm thử giao diện người dùng (UI Testing/End-to-End Testing):**
  - Sử dụng các công cụ như Cypress, Playwright, Selenium để mô phỏng hành vi người dùng trên trình duyệt. **Công cụ:** Cypress, Playwright, Selenium WebDriver.
  - Kiểm tra các kịch bản người dùng quan trọng (critical user journeys) để đảm bảo toàn bộ ứng dụng hoạt động như mong đợi. **Cách kiểm tra:** Viết các kịch bản tự động hóa mô phỏng hành vi người dùng thật.
  - Kiểm tra hiển thị trên các trình duyệt và thiết bị khác nhau (cross-browser and cross-device compatibility). **Công cụ:** BrowserStack, Sauce Labs, LambdaTest.
  - Kiểm tra các trạng thái loading, error, empty state của UI. **Cách kiểm tra:** Mô phỏng các trạng thái này từ backend hoặc bằng cách thay đổi dữ liệu test.
- **Kiểm thử hiệu năng (Performance Testing):**
  - Đo lường thời gian tải trang (page load time), thời gian tương tác (Time to Interactive). **Công cụ:** Google Lighthouse, WebPageTest, Chrome DevTools.
  - Kiểm tra hiệu suất của các animation và chuyển động trên UI. **Cách kiểm tra:** Sử dụng Chrome DevTools để phân tích hiệu suất rendering.
  - Sử dụng các công cụ như Lighthouse để đánh giá hiệu suất, SEO, accessibility. **Công cụ:** Google Lighthouse.
- **Kiểm thử khả năng truy cập (Accessibility Testing):**
  - Đảm bảo ứng dụng có thể được sử dụng bởi người dùng khuyết tật (ví dụ: sử dụng bàn phím, trình đọc màn hình). **Công cụ:** Axe, Pa11y, Lighthouse (Accessibility Audit).
  - Tuân thủ các tiêu chuẩn WCAG (Web Content Accessibility Guidelines). **Cách kiểm tra:** Sử dụng các công cụ tự động và kiểm tra thủ công theo hướng dẫn WCAG.
- **Kiểm thử bảo mật (Security Testing):**
  - Kiểm tra các lỗ hổng XSS, CSRF (Cross-Site Request Forgery). **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite

(https://portswigger.net/burp/communitydownload).

- Đảm bảo việc lưu trữ token (JWT) an toàn (ví dụ: sử dụng HttpOnly cookies). **Cách kiểm tra:** Kiểm tra cách token được lưu trữ và truyền đi.
- Kiểm tra việc xử lý dữ liệu nhạy cảm trên client-side. **Cách kiểm tra:** Kiểm tra xem dữ liệu nhạy cảm có được hiển thị hoặc lưu trữ không an toàn trên trình duyệt không.

7. Admin Panel (Nếu tách riêng)

Mục đích: Cung cấp giao diện quản trị riêng biệt cho phép quản trị viên quản lý hệ thống.

7.1. Tính năng

- **Giao diện quản trị chuyên biệt:** Cung cấp giao diện riêng biệt, tập trung vào các chức năng quản lý và giám sát hệ thống cho quản trị viên.
- **Dashboard tổng quan:** Hiển thị các số liệu thống kê quan trọng, biểu đồ, và thông tin tổng quan về tình trạng hệ thống (số lượng người dùng, giao dịch, lỗi).
- **Quản lý người dùng chi tiết:** Chức năng tìm kiếm, lọc, xem, chỉnh sửa, khóa/mở khóa tài khoản người dùng, thay đổi vai trò.
- **Quản lý giao dịch chi tiết:** Chức năng tìm kiếm, lọc, xem chi tiết các giao dịch, có thể bao gồm chức năng hoàn tiền hoặc điều chỉnh giao dịch.
- **Quản lý nội dung:** Giao diện để quản trị viên tạo, chỉnh sửa, xóa các trang tĩnh, FAQ, thông báo.
- **Quản lý dịch vụ/gói:** Giao diện để quản lý các dịch vụ toán học, gói nạp tiền, giá cả.
- **Quản lý quyền hạn:** Giao diện để quản lý các tài khoản quản trị viên, phân quyền truy cập vào các chức năng khác nhau của Admin Panel.
- **Xem log hệ thống:** Cho phép quản trị viên xem các log hoạt động của hệ thống và người dùng để phục vụ mục đích kiểm toán và gỡ lỗi.
- **Cấu hình hệ thống:** Giao diện để quản trị viên thay đổi các cấu hình chung của hệ thống (ví dụ: tỷ giá, thông báo).

7.2. Phương pháp kiểm tra

- **Kiểm thử đơn vị (Unit Testing):**
  - Kiểm tra các component UI và logic nghiệp vụ riêng của Admin Panel. **Công cụ:** Jest, React Testing Library.
  - Kiểm tra các hàm gọi API đến Admin Service. **Công cụ:** Jest, React Testing Library, Mock Service Worker (MSW).

- **Kiểm thử tích hợp (Integration Testing):**
  - Kiểm tra luồng dữ liệu từ Admin Service đến Admin Panel và ngược lại. **Công cụ:** Cypress, Playwright.
  - Kiểm tra các kịch bản quản trị viên (ví dụ: khóa tài khoản người dùng, cập nhật nội dung). **Cách kiểm tra:** Viết các kịch bản tự động hóa mô phỏng hành vi quản trị viên.
- **Kiểm thử giao diện người dùng (UI Testing/End-to-End Testing):**
  - Sử dụng các công cụ tự động hóa để kiểm tra các chức năng quản trị quan trọng. **Công cụ:** Cypress, Playwright, Selenium WebDriver.
  - Kiểm tra tính đúng đắn của dữ liệu hiển thị trên dashboard và các bảng quản lý. **Cách kiểm tra:** So sánh dữ liệu hiển thị với dữ liệu mong đợi từ backend.
  - Kiểm tra phân quyền hiển thị các chức năng dựa trên vai trò của quản trị viên. **Cách kiểm tra:** Đăng nhập bằng các tài khoản quản trị viên có vai trò khác nhau và kiểm tra các chức năng được hiển thị.
- **Kiểm thử bảo mật (Security Testing):**
  - Đảm bảo chỉ quản trị viên được ủy quyền mới có thể truy cập Admin Panel. **Công cụ:** OWASP ZAP (https://www.zaproxy.org/download/), Burp Suite (https://portswigger.net/burp/communitydownload).
  - Kiểm tra các lỗ hổng XSS, CSRF, và các vấn đề bảo mật khác liên quan đến giao diện web. **Cách kiểm tra:** Chèn các script độc hại, thử tấn công CSRF.
  - Đảm bảo các thao tác nhạy cảm yêu cầu xác thực lại hoặc có cơ chế bảo vệ bổ sung. **Cách kiểm tra:** Kiểm tra xem các thao tác như xóa người dùng có yêu cầu xác nhận lại mật khẩu không.
- **Kiểm thử hiệu năng (Performance Testing):**
  - Đánh giá thời gian tải của các trang có nhiều dữ liệu (ví dụ: danh sách người dùng, lịch sử giao dịch). **Công cụ:** Google Lighthouse, WebPageTest, Chrome DevTools.
  - Kiểm tra hiệu suất của các chức năng tìm kiếm, lọc, phân trang. **Cách kiểm tra:** Thực hiện các thao tác này với lượng dữ liệu lớn và đo thời gian phản hồi.

8. API Gateway

Mục đích: Định tuyến, cân bằng tải, xác thực, và quản lý truy cập đến các Microservice Backend.

8.1. Tính năng

- **Định tuyến yêu cầu:** Chuyển tiếp các yêu cầu từ Frontend đến đúng Microservice Backend dựa trên các quy tắc định tuyến.

- **Cân bằng tải:** Phân phối các yêu cầu đến các instance khác nhau của một Microservice để đảm bảo hiệu suất và khả năng chịu lỗi.
- **Xác thực và ủy quyền:** Thực hiện xác thực JWT và ủy quyền ở cấp độ gateway, giảm tải cho từng Microservice.
- **Giới hạn tốc độ (Rate Limiting):** Kiểm soát số lượng yêu cầu mà một client có thể gửi trong một khoảng thời gian nhất định để ngăn chặn tấn công DDoS và lạm dụng API.
- **Ghi log và giám sát:** Ghi lại các yêu cầu và phản hồi đi qua gateway để phục vụ mục đích giám sát và gỡ lỗi.
- **Chuyển đổi giao thức:** Chuyển đổi các giao thức (ví dụ: HTTP sang gRPC) nếu cần.
- **Caching:** Lưu trữ các phản hồi thường xuyên được yêu cầu để giảm tải cho các Microservice Backend.
- **Xử lý lỗi tập trung:** Cung cấp các phản hồi lỗi nhất quán cho client khi có lỗi xảy ra ở bất kỳ Microservice nào.

8.2. Phương pháp kiểm tra

- **Kiểm thử định tuyến:**
  - Gửi các yêu cầu đến các endpoint khác nhau của gateway và xác minh rằng chúng được định tuyến đến đúng Microservice Backend. **Công cụ:** Postman (https://www.postman.com/downloads/), cURL (https://curl.se/), `requests` (https://requests.readthedocs.io/) (Python), `httpx` (https://www.python-httpx.org/) (Python).
  - Kiểm tra các quy tắc định tuyến phức tạp (ví dụ: định tuyến dựa trên header, query parameter). **Cách kiểm tra:** Cấu hình các yêu cầu với các header hoặc query parameter khác nhau và kiểm tra đích đến.
- **Kiểm thử cân bằng tải:**
  - Triển khai nhiều instance của một Microservice và gửi tải cao đến gateway để xác minh rằng các yêu cầu được phân phối đều giữa các instance. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
  - Kiểm tra hành vi khi một instance bị lỗi hoặc không khả dụng. **Cách kiểm tra:** Tắt một instance của microservice và kiểm tra xem gateway có tự động chuyển hướng lưu lượng truy cập sang các instance còn lại không.
- **Kiểm thử xác thực và ủy quyền:**
  - Gửi các yêu cầu với JWT hợp lệ, không hợp lệ, hết hạn để xác minh rằng gateway xử lý xác thực đúng cách. **Công cụ:** Postman (https://www.postman.com/downloads/),

cURL (https://curl.se/), thư viện JWT (ví dụ: `PyJWT` trong Python) để tạo các token test.

- Kiểm tra các yêu cầu không có token hoặc token không đúng định dạng. **Cách kiểm tra:** Gửi yêu cầu mà không có header Authorization hoặc với token bị lỗi.
- Kiểm tra các quy tắc ủy quyền (ví dụ: chỉ cho phép admin truy cập một số API nhất định). **Cách kiểm tra:** Thử truy cập các API yêu cầu quyền Admin bằng token của người dùng thông thường.

- **Kiểm thử giới hạn tốc độ (Rate Limiting):**
  - Gửi một lượng lớn yêu cầu vượt quá giới hạn tốc độ đã cấu hình và xác minh rằng gateway trả về lỗi 429 Too Many Requests. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
- **Kiểm thử ghi log và giám sát:**
  - Kiểm tra xem các log yêu cầu và phản hồi có được ghi lại đầy đủ và chính xác trong hệ thống giám sát. **Công cụ:** Kibana (cho ELK Stack), Grafana (cho Prometheus).
- **Kiểm thử khả năng phục hồi (Resilience Testing):**
  - Kiểm tra hành vi của gateway khi một hoặc nhiều Microservice Backend không phản hồi hoặc gặp lỗi. **Công cụ:** Chaos Monkey (https://github.com/Netflix/chaosmonkey), Gremlin (https://www.gremlin.com/), hoặc mô phỏng lỗi thủ công.
  - Đảm bảo gateway trả về các phản hồi lỗi phù hợp và không làm sập toàn bộ hệ thống. **Cách kiểm tra:** Gây lỗi cho microservice backend và kiểm tra phản hồi lỗi của gateway.
- **Kiểm thử hiệu năng:**
  - Đánh giá thông lượng và độ trễ của gateway dưới tải cao. **Công cụ:** Apache JMeter, Locust (https://locust.io/), k6 (https://k6.io/).
  - Xác định khả năng chịu tải tối đa của gateway. **Cách kiểm tra:** Tăng dần tải lên gateway cho đến khi hiệu suất giảm đáng kể hoặc xảy ra lỗi.

9. DevOps Components

Các thành phần DevOps đóng vai trò quan trọng trong việc tự động hóa quá trình phát triển, triển khai, và vận hành hệ thống. Mặc dù không phải là các microservice theo nghĩa truyền thống, chúng vẫn có các tính năng và cần được kiểm tra kỹ lưỡng.

9.1. Docker Compose (https://docs.docker.com/compose/)

Mục đích: Định nghĩa và chạy các ứng dụng Docker đa container cho môi trường phát triển/kiểm thử cục bộ.

9.1.1. Tính năng

- **Khởi tạo môi trường cục bộ:** Cho phép nhà phát triển dễ dàng khởi động toàn bộ hoặc một phần hệ thống microservice trên máy cục bộ với một lệnh duy nhất.
- **Quản lý phụ thuộc:** Định nghĩa các dịch vụ (microservice, database, message broker) và các phụ thuộc giữa chúng.
- **Cấu hình môi trường:** Thiết lập các biến môi trường, port mapping, volume mounting cho từng dịch vụ.
- **Mở rộng dịch vụ:** Cho phép dễ dàng mở rộng số lượng instance của một dịch vụ để kiểm thử cục bộ.

9.1.2. Phương pháp kiểm tra

- **Kiểm thử khởi động:**
  - Chạy `docker-compose up` và xác minh rằng tất cả các dịch vụ được định nghĩa đều khởi động thành công và không có lỗi. **Cách kiểm tra:** Quan sát output của `docker-compose up`, kiểm tra trạng thái của các container bằng `docker ps` và xem logs bằng `docker logs <container_name>`.
  - Kiểm tra log của từng container để đảm bảo chúng hoạt động đúng. **Cách kiểm tra:** Sử dụng `docker logs <container_name>`.
- **Kiểm thử kết nối:**
  - Đảm bảo các microservice có thể giao tiếp với nhau và với database, message broker như đã cấu hình. **Cách kiểm tra:** Sử dụng `docker exec -it <container_name> bash` để vào trong container và dùng `ping`, `curl` hoặc các lệnh kết nối database/message broker để kiểm tra.
  - Sử dụng các lệnh `docker exec` để vào trong container và kiểm tra kết nối mạng. **Công cụ:** `docker exec`, `ping`, `curl`.
- **Kiểm thử cấu hình:**
  - Xác minh rằng các biến môi trường, port mapping, volume mounting được áp dụng chính xác cho từng dịch vụ. **Cách kiểm tra:** Kiểm tra biến môi trường bên trong container bằng `printenv`, kiểm tra port mapping bằng `docker port <container_name>`, kiểm tra volume mounting bằng cách tạo/sửa file trong volume và kiểm tra từ host.
- **Kiểm thử khả năng mở rộng:**
  - Sử dụng `docker-compose up --scale <service>=<count>` để kiểm tra khả năng mở rộng cục bộ và đảm bảo các instance mới hoạt động đúng. **Cách kiểm tra:** Tăng số lượng

instance và kiểm tra xem tất cả các instance có khởi động và hoạt động bình thường không.

9.2. Kubernetes (https://kubernetes.io/) Manifests

Mục đích: Định nghĩa cách triển khai, quản lý và mở rộng các ứng dụng trên cụm Kubernetes (https://kubernetes.io/) cho môi trường production.

9.2.1. Tính năng

- **Triển khai ứng dụng:** Định nghĩa các Deployment để triển khai các container của microservice.
- **Phơi bày dịch vụ:** Định nghĩa các Service để expose các Pod và cho phép giao tiếp nội bộ/bên ngoài cụm.
- **Quản lý truy cập bên ngoài:** Định nghĩa Ingress để quản lý truy cập HTTP/HTTPS từ bên ngoài vào các dịch vụ.
- **Quản lý cấu hình và bí mật:** Lưu trữ cấu hình không nhạy cảm (ConfigMaps) và thông tin nhạy cảm (Secrets) một cách an toàn.
- **Tự động mở rộng:** Cấu hình Horizontal Pod Autoscaler (HPA) để tự động mở rộng số lượng Pod dựa trên tải.
- **Cập nhật cuốn chiếu (Rolling Updates):** Cho phép cập nhật ứng dụng mà không làm gián đoạn dịch vụ.

9.2.2. Phương pháp kiểm tra

- **Kiểm thử cú pháp:**
  - Sử dụng `kubectl apply --dry-run -f <file>` hoặc các công cụ linting (ví dụ: `kubeval`, `conftest`) để kiểm tra cú pháp và tính hợp lệ của các manifest. **Công cụ:** `kubectl` (https://kubernetes.io/docs/tasks/tools/install-kubectl/), `kubeval`, `conftest`.
- **Kiểm thử triển khai:**
  - Triển khai các manifest lên một cụm Kubernetes (https://kubernetes.io/) thử nghiệm và xác minh rằng các Pod, Deployment, Service, Ingress được tạo thành công. **Cách kiểm tra:** Sử dụng `kubectl get pods`, `kubectl get deployments`, `kubectl get services`, `kubectl get ingress` và kiểm tra trạng thái của chúng.
  - Kiểm tra trạng thái của các Pod, đảm bảo chúng đang chạy và sẵn sàng. **Cách kiểm tra:** `kubectl describe pod <pod_name>`, `kubectl logs <pod_name>`.
- **Kiểm thử kết nối:**
  - Kiểm tra kết nối nội bộ giữa các Pod và Service trong cụm. **Cách kiểm tra:** Sử dụng `kubectl exec` vào một Pod và dùng `curl` hoặc `ping` đến các Service.

    - Kiểm tra truy cập từ bên ngoài thông qua Ingress. **Cách kiểm tra:** Truy cập URL của Ingress từ trình duyệt hoặc `curl`.
- **Kiểm thử cấu hình và bí mật:**
    - Xác minh rằng các ConfigMaps và Secrets được mount đúng cách vào các Pod và các ứng dụng có thể đọc được chúng. **Cách kiểm tra:** `kubectl exec` vào Pod và kiểm tra nội dung của các file được mount từ ConfigMap/Secret.
- **Kiểm thử cập nhật:**
    - Thực hiện cập nhật ứng dụng bằng cách thay đổi image trong Deployment và quan sát quá trình rolling update để đảm bảo không có downtime. **Cách kiểm tra:** Theo dõi trạng thái của Deployment bằng `kubectl rollout status deployment/<deployment_name>` và kiểm tra tính khả dụng của ứng dụng trong quá trình cập nhật.
- **Kiểm thử tự động mở rộng:**
    - Tạo tải lên một dịch vụ và xác minh rằng HPA hoạt động đúng, tự động tăng/giảm số lượng Pod. **Công cụ:** `hey`, `ApacheBench` để tạo tải. **Cách kiểm tra:** Theo dõi số lượng Pod bằng `kubectl get hpa` và `kubectl get pods` khi tải tăng/giảm.

9.3. Monitoring (Prometheus, Grafana, ELK Stack)

Mục đích: Giám sát hiệu suất, thu thập log, và trực quan hóa dữ liệu để đảm bảo hệ thống hoạt động ổn định và phát hiện sự cố kịp thời.

9.3.1. Tính năng

- **Thu thập Metrics:** Prometheus thu thập các chỉ số hiệu suất (CPU, RAM, network, số lượng request, thời gian phản hồi) từ các microservice và hạ tầng.
- **Trực quan hóa dữ liệu:** Grafana tạo các dashboard tùy chỉnh để hiển thị các metrics một cách trực quan, giúp dễ dàng theo dõi tình trạng hệ thống.
- **Thu thập và phân tích Log:** ELK Stack (Elasticsearch, Logstash, Kibana) thu thập, xử lý, lưu trữ và cho phép tìm kiếm, phân tích các log từ tất cả các thành phần của hệ thống.
- **Cảnh báo (Alerting):** Cấu hình các quy tắc cảnh báo dựa trên metrics hoặc log để thông báo khi có sự cố hoặc ngưỡng vượt quá.

9.3.2. Phương pháp kiểm tra

- **Kiểm thử thu thập Metrics:**
    - Đảm bảo Prometheus có thể scrape metrics từ tất cả các microservice và các thành phần hạ tầng (Node Exporter, cAdvisor). **Cách kiểm tra:** Truy cập giao diện

Prometheus UI và kiểm tra trạng thái của các target, xem các metrics có được thu thập đúng không.

- Kiểm tra các metrics quan trọng có được thu thập đúng và đầy đủ. **Công cụ:** Prometheus UI, PromQL (Prometheus Query Language).

- **Kiểm thử Dashboard Grafana:**
  - Truy cập các dashboard Grafana và xác minh rằng dữ liệu được hiển thị chính xác và cập nhật theo thời gian thực. **Cách kiểm tra:** So sánh dữ liệu trên dashboard với dữ liệu thực tế hoặc từ Prometheus.
  - Kiểm tra các biểu đồ, bảng có hoạt động đúng và cung cấp thông tin hữu ích. **Công cụ:** Grafana UI.
- **Kiểm thử thu thập Log:**
  - Tạo ra các log từ các microservice và xác minh rằng chúng được thu thập bởi Logstash, lưu trữ trong Elasticsearch và có thể tìm kiếm được trong Kibana. **Cách kiểm tra:** Gửi các log test từ ứng dụng và tìm kiếm chúng trong Kibana.
  - Kiểm tra các trường log (fields) có được parse đúng cách. **Công cụ:** Kibana Discover.
- **Kiểm thử cảnh báo (Alerting):**
  - Cấu hình các quy tắc cảnh báo dựa trên metrics hoặc log để thông báo khi có sự cố hoặc ngưỡng vượt quá. **Công cụ:** Alertmanager (cho Prometheus), Kibana Alerting.
  - Kích hoạt một tình huống giả lập để gây ra cảnh báo (ví dụ: tăng tải đột ngột, tắt một dịch vụ) và xác minh rằng cảnh báo được gửi đến kênh thông báo đã cấu hình (email, Slack). **Cách kiểm tra:** Gây lỗi có chủ đích và kiểm tra xem cảnh báo có được gửi đi không, nội dung cảnh báo có chính xác không.

9.4. Scripts (CI/CD, Quản lý, Backup)

Mục đích: Tự động hóa các tác vụ lặp đi lặp lại trong quá trình phát triển, triển khai, vận hành và bảo trì.

9.4.1. Tính năng

- **Tự động hóa CI/CD:** Các script để tự động hóa quá trình build, test, đóng gói (Docker image), và triển khai ứng dụng.
- **Quản lý môi trường:** Script để thiết lập môi trường phát triển, staging, production.
- **Quản lý database:** Script để thực hiện migration database, backup và restore dữ liệu.
- **Quản lý log và monitoring:** Script để quản lý các công cụ log và monitoring, ví dụ như xoay log, dọn dẹp dữ liệu cũ.
- **Tự động hóa backup:** Script để tự động sao lưu dữ liệu và cấu hình hệ thống.

9.4.2. Phương pháp kiểm tra

- **Kiểm thử script CI/CD:**
  - Chạy pipeline CI/CD trên một nhánh thử nghiệm và xác minh rằng tất cả các bước (build, test, đóng gói (Docker image), và triển khai ứng dụng) đều thành công. **Công cụ:** Jenkins, GitLab CI/CD, GitHub Actions, CircleCI. **Cách kiểm tra:** Theo dõi log của pipeline, kiểm tra các artifact được tạo ra (Docker images, gói triển khai).
  - Kiểm tra các Docker image được build đúng cách và được push lên registry. **Cách kiểm tra:** Kéo image từ registry và chạy thử, kiểm tra nội dung image.
  - Kiểm tra ứng dụng được triển khai thành công lên môi trường đích. **Cách kiểm tra:** Truy cập ứng dụng đã triển khai, kiểm tra các endpoint, hoặc sử dụng các công cụ kiểm thử tự động.
- **Kiểm thử script quản lý môi trường:**
  - Chạy script thiết lập môi trường trên một máy mới và xác minh rằng tất cả các công cụ và phụ thuộc cần thiết được cài đặt đúng. **Cách kiểm tra:** Kiểm tra phiên bản của các công cụ, kiểm tra các thư mục, file cấu hình được tạo ra.
- **Kiểm thử script quản lý database:**
  - Chạy script migration trên một database thử nghiệm và xác minh rằng schema được cập nhật đúng. **Cách kiểm tra:** Kiểm tra schema của database sau khi chạy migration, kiểm tra dữ liệu mẫu (nếu có).
  - Thực hiện backup và restore dữ liệu trên môi trường thử nghiệm để đảm bảo dữ liệu có thể được phục hồi thành công. **Cách kiểm tra:** Tạo dữ liệu mẫu, chạy backup, xóa dữ liệu, chạy restore và kiểm tra lại dữ liệu.
- **Kiểm thử script backup:**
  - Chạy script backup và xác minh rằng các file backup được tạo ra và lưu trữ ở vị trí mong muốn. **Cách kiểm tra:** Kiểm tra sự tồn tại, kích thước, và nội dung của các file backup.
  - Thực hiện restore từ các file backup này để đảm bảo tính toàn vẹn của dữ liệu. **Cách kiểm tra:** Thực hiện restore trên một môi trường riêng biệt và kiểm tra tính toàn vẹn của dữ liệu đã phục hồi.
- **Kiểm thử lỗi:**
  - Kiểm tra hành vi của các script khi gặp lỗi (ví dụ: thiếu quyền, thiếu file, lỗi mạng) và đảm bảo chúng xử lý lỗi một cách graceful và cung cấp thông báo lỗi rõ ràng. **Cách kiểm tra:** Mô phỏng các tình huống lỗi và kiểm tra output của script, mã thoát (exit code).

10. Đánh giá và Đề xuất Giải pháp Giao diện Kiểm thử

Việc tạo ra một giao diện chuyên dụng cho việc kiểm thử (Testing Dashboard/Interface) là một ý tưởng rất đáng cân nhắc, đặc biệt trong một hệ thống microservices phức tạp. Giao diện này có thể mang lại nhiều lợi ích trong việc tự động hóa, trực quan hóa và đơn giản hóa quy trình kiểm thử.

10.1. Lợi ích của Giao diện Kiểm thử

- **Trực quan hóa:** Cung cấp một cái nhìn tổng quan, trực quan về trạng thái của các dịch vụ, kết quả của các bài kiểm thử (pass/fail), và các chỉ số hiệu năng. Điều này giúp đội ngũ phát triển và QA nhanh chóng nắm bắt được tình hình sức khỏe của hệ thống.
- **Đơn giản hóa:** Cho phép người dùng không chuyên về kỹ thuật (ví dụ: QA, Product Manager) có thể thực hiện các kịch bản kiểm thử phức tạp mà không cần phải chạy các lệnh hoặc script phức tạp.
- **Tập trung hóa:** Gom tất cả các công cụ và kết quả kiểm thử vào một nơi duy nhất, thay vì phải truy cập nhiều hệ thống khác nhau (Jenkins, Postman (https://www.postman.com/downloads/), Grafana, Kibana).
- **Tăng tốc độ phản hồi:** Giúp nhanh chóng xác định được microservice nào đang gặp lỗi khi một bài kiểm thử thất bại, từ đó rút ngắn thời gian gỡ lỗi.
- **Hỗ trợ kiểm thử thủ công:** Cung cấp một môi trường thân thiện để thực hiện các bài kiểm thử thủ công (manual testing) một cách có hệ thống và dễ dàng ghi nhận kết quả.

10.2. Nhược điểm và Thách thức

- **Chi phí phát triển:** Xây dựng và duy trì một giao diện kiểm thử riêng biệt đòi hỏi thời gian và nguồn lực phát triển đáng kể.
- **Phức tạp trong tích hợp:** Cần phải tích hợp với nhiều công cụ và hệ thống khác nhau (CI/CD, các công cụ kiểm thử, hệ thống giám sát), điều này có thể phức tạp và tốn thời gian.
- **Rủi ro trở thành một sản phẩm phụ:** Giao diện kiểm thử có thể trở thành một sản phẩm phần mềm phức tạp cần được quản lý, cập nhật và bảo trì riêng.

10.3. Các Giải pháp Đề xuất

Dựa trên các lợi ích và thách thức, có một số hướng tiếp cận để xây dựng giao diện kiểm thử, từ đơn giản đến phức tạp:

Giải pháp 1: Tận dụng và Mở rộng các Công cụ Hiện có (Khuyến nghị)

Đây là giải pháp ít tốn kém và hiệu quả nhất trong giai đoạn đầu.

- **Trực quan hóa kết quả kiểm thử:**
  - **Công cụ:** Tích hợp các plugin báo cáo vào các công cụ CI/CD như Jenkins (ví dụ: HTML Publisher Plugin, Test Results Analyzer Plugin) hoặc GitLab CI/CD (sử dụng tính năng Test Reports).
  - **Cách thực hiện:** Cấu hình các pipeline CI/CD để sau khi chạy các bài kiểm thử (pytest, Jest, Cypress), chúng sẽ tạo ra các báo cáo trực quan (HTML, XML). Các báo cáo này sẽ được hiển thị trực tiếp trên giao diện của Jenkins/GitLab, cho phép xem kết quả, log lỗi, và ảnh chụp màn hình (đối với Cypress).
- **Kích hoạt kiểm thử thủ công:**
  - **Công cụ:** Sử dụng tính năng "Build with Parameters" của Jenkins hoặc "Run pipeline" của GitLab CI/CD.
  - **Cách thực hiện:** Tạo các pipeline cho phép người dùng nhập các tham số (ví dụ: tên kịch bản kiểm thử, môi trường, phiên bản ứng dụng) và kích hoạt chạy các bài kiểm thử tương ứng. Điều này cho phép QA có thể chạy lại các bài kiểm thử cụ thể mà không cần can thiệp vào code.
- **Dashboard giám sát:**
  - **Công cụ:** Tận dụng tối đa Grafana và Kibana.
  - **Cách thực hiện:** Tạo một dashboard Grafana chuyên dụng cho việc kiểm thử, hiển thị các metrics liên quan đến hiệu năng (thời gian phản hồi, tỷ lệ lỗi) và tài nguyên hệ thống trong quá trình chạy kiểm thử. Sử dụng Kibana để tạo các dashboard trực quan hóa log, giúp dễ dàng tìm kiếm và phân tích lỗi.

Ưu điểm:

- Chi phí thấp, tận dụng được các công cụ đã có.
- Triển khai nhanh chóng.
- Dễ dàng bảo trì vì không phải phát triển một ứng dụng mới.

Nhược điểm:

- Giao diện phân mảnh, người dùng phải truy cập nhiều công cụ khác nhau.
- Có thể không hoàn toàn thân thiện với người dùng không chuyên về kỹ thuật.

Giải pháp 2: Xây dựng một Giao diện Kiểm thử Đơn giản (Internal Tool)

Nếu giải pháp 1 không đáp ứng đủ nhu cầu, có thể xem xét xây dựng một công cụ nội bộ đơn giản.

- **Công nghệ:** Sử dụng một framework web nhẹ như Flask (Python) hoặc Express (Node.js) cho backend và React/Vue.js cho frontend.

- **Tính năng cốt lõi:**
  - **Danh sách các kịch bản kiểm thử:** Hiển thị danh sách các bài kiểm thử (API test, E2E test) có thể được thực thi.
  - **Nút "Run Test":** Cho phép người dùng kích hoạt một bài kiểm thử cụ thể bằng cách gọi API của Jenkins/GitLab CI/CD.
  - **Hiển thị kết quả:** Hiển thị trạng thái (pass/fail) và log của các lần chạy kiểm thử gần nhất.
  - **Liên kết đến các công cụ khác:** Cung cấp các liên kết nhanh đến các báo cáo chi tiết trên Jenkins, dashboard Grafana, hoặc log trên Kibana.

Ưu điểm:

- Giao diện tập trung, thân thiện hơn với người dùng.
- Tùy chỉnh được theo nhu cầu cụ thể của dự án.

Nhược điểm:

- Tốn chi phí phát triển và bảo trì.
- Cần có đội ngũ riêng để phát triển và vận hành.

Giải pháp 3: Sử dụng các Nền tảng Quản lý Kiểm thử (Test Management Platforms)

Đối với các dự án lớn và yêu cầu quy trình QA chuyên nghiệp, có thể xem xét sử dụng các nền tảng quản lý kiểm thử thương mại hoặc mã nguồn mở.

- **Công cụ:** TestRail, Zephyr, qTest (thương mại), TestLink, Kiwi TCMS (mã nguồn mở).
- **Tính năng:**
  - Quản lý các kịch bản kiểm thử (test cases).
  - Lập kế hoạch và thực thi các chu kỳ kiểm thử (test cycles).
  - Tích hợp với các công cụ tự động hóa (Jenkins, Cypress) để tự động cập nhật kết quả.
  - Tạo báo cáo và thống kê chi tiết về chất lượng sản phẩm.

Ưu điểm:

- Cung cấp một quy trình quản lý kiểm thử chuyên nghiệp và toàn diện.
- Nhiều tính năng mạnh mẽ, báo cáo chi tiết.

Nhược điểm:

- Chi phí cao (đối với các công cụ thương mại).

- Cần thời gian để học và tích hợp vào quy trình hiện có.

10.4. Kết luận và Đề xuất

Đề xuất:

1. **Bắt đầu với Giải pháp 1:** Tận dụng và mở rộng các công cụ hiện có (Jenkins/GitLab, Grafana, Kibana). Đây là cách tiếp cận thực tế và hiệu quả nhất để bắt đầu. Cần tập trung vào việc xây dựng các pipeline CI/CD mạnh mẽ và các dashboard giám sát hữu ích.
2. **Đánh giá lại sau một thời gian:** Sau khi hệ thống đã vận hành ổn định và quy trình kiểm thử đã vào guồng, đánh giá lại xem giải pháp 1 có đáp ứng đủ nhu cầu không. Nếu có các yêu cầu phức tạp hơn hoặc cần một giao diện thân thiện hơn cho đội ngũ QA, lúc đó có thể cân nhắc **xây dựng một công cụ nội bộ đơn giản (Giải pháp 2)**.
3. **Giải pháp 3** chỉ nên được xem xét khi dự án có quy mô rất lớn, đội ngũ QA chuyên biệt, và yêu cầu một quy trình quản lý kiểm thử chặt chẽ theo các tiêu chuẩn quốc tế.

Việc có một giao diện kiểm thử là một mục tiêu dài hạn đáng giá. Tuy nhiên, trong giai đoạn hiện tại, việc tập trung vào việc xây dựng nền tảng kiểm thử tự động vững chắc và tận dụng hiệu quả các công cụ CI/CD và giám sát hiện có sẽ mang lại lợi ích thiết thực và nhanh chóng hơn.

Công cụ: Docker Desktop (https://www.docker.com/products/docker-desktop/) (https://www.docker.com/products/docker-desktop/) (bao gồm Docker Engine và Docker Compose).

Công cụ: `minikube ([https://minikube.sigs.k8s.io/](https://minikube.sigs.k8s.io/))` (https://minikube (https://minikube.sigs.k8s.io/).sigs.k8s.io/) cho môi trường phát triển cục bộ.