Đề cương chi tiết phát triển Website dịch vụ toán học

1. Giới thiệu

Đề cương này trình bày kế hoạch chi tiết để phát triển một website dịch vụ toán học toàn diện, cho phép người dùng nạp tiền và sử dụng các dịch vụ giải toán trực tuyến. Website được thiết kế với **kiến trúc Microservices**, đặc biệt chú trọng đến khả năng mở rộng để dễ dàng tích hợp các ứng dụng và dịch vụ mới trong tương lai.

1.1. Mục tiêu dự án

Xây dựng một nền tảng kinh doanh vững chắc cung cấp các dịch vụ giải toán trực tuyến với các tính năng chính:

- Hệ thống nạp tiền và thanh toán tích hợp
- Nhiều dịch vụ toán học đa dạng (giải phương trình bậc 2, hệ phương trình, v.v.)
- Giao diện người dùng thân thiện và responsive
- Hệ thống quản trị hoàn chỉnh
- Kiến trúc Microservices có khả năng mở rộng cao

1.2. Phạm vi dự án

Phạm vi ban đầu:

- Website với đầy đủ chức năng: đăng ký/đăng nhập, nạp tiền, sử dụng dịch vụ, quản lý tài khoản
- Ít nhất 2 dịch vụ toán học: Giải phương trình bậc 2 và Giải hệ phương trình
- Hệ thống quản trị cho quản lý người dùng, dịch vụ và giao dịch
- Tích hợp cổng thanh toán
- Thiết kế kiến trúc Microservices cho phép mở rộng dễ dàng

Khả năng mở rộng tương lai:

- Thêm các dịch vụ toán học mới (các phần "n-...") một cách độc lập như các microservice riêng biệt
- Mở rộng ra thị trường quốc tế
- Tích hợp AI/ML cho các bài toán phức tạp

2. Phân tích yêu cầu và thiết kế kiến trúc tổng thể

2.1. Yêu cầu chức năng chi tiết

2.1.1. Quản lý người dùng (User Management):

- **Đăng ký tài khoản:** Cho phép người dùng mới tạo tài khoản bằng email/số điện thoại và mật khẩu. Hệ thống sẽ kiểm tra tính hợp lệ của email/số điện thoại và gửi mã xác thực để hoàn tất đăng ký. Mật khẩu sẽ được mã hóa bằng thuật toán bcrypt trước khi lưu vào cơ sở dữ liệu.
- **Đăng nhập/Đăng xuất:** Cung cấp cơ chế xác thực an toàn bằng JWT (JSON Web Token). Sau khi đăng nhập thành công, hệ thống sẽ trả về một access token và một refresh token. Access token sẽ được sử dụng cho các yêu cầu tiếp theo, trong khi refresh token sẽ được dùng để lấy access token mới khi hết hạn.
- **Quản lý hồ sơ cá nhân:** Người dùng có thể xem và cập nhật thông tin cá nhân (tên, email, số điện thoại, địa chỉ), thay đổi mật khẩu. Mọi thay đổi đều yêu cầu xác thực lại mật khẩu hiện tại.
- **Quản lý số dư tài khoản:** Hiển thị số dư hiện tại, lịch sử nạp tiền và lịch sử sử dụng dịch vụ chi tiết. Lịch sử giao dịch sẽ bao gồm thông tin về thời gian, số tiền, loại giao dịch và trạng thái.
- **Khôi phục mật khẩu:** Chức năng quên mật khẩu qua email hoặc số điện thoại. Hệ thống sẽ gửi một liên kết hoặc mã OTP để người dùng đặt lại mật khẩu mới.

2.1.2. Quản lý dịch vụ (Service Management):

- **Danh mục dịch vụ:** Hiển thị danh sách tất cả các dịch vụ giải toán hiện có với mô tả chi tiết, ví dụ minh họa và mức phí. Các dịch vụ sẽ được phân loại theo chủ đề (ví dụ: Đại số, Giải tích, Thống kê) để người dùng dễ dàng tìm kiếm.
- **Sử dụng dịch vụ:** Cung cấp giao diện trực quan để người dùng nhập dữ liệu đầu vào cho từng bài toán và nhận kết quả đầu ra. Giao diện sẽ được thiết kế riêng biệt cho từng loại bài toán để đảm bảo tính dễ sử dụng. Kết quả sẽ được hiển thị rõ ràng, kèm theo các bước giải chi tiết (nếu có).
- **Lịch sử sử dụng dịch vụ:** Lưu trữ và hiển thị chi tiết các lần người dùng đã sử dụng dịch vụ, bao gồm loại dịch vụ, thời gian, chi phí và kết quả (hoặc link đến kết quả). Người dùng có thể xem lại lịch sử sử dụng dịch vụ của mình bất cứ lúc nào.

2.1.3. Hệ thống nạp tiền và thanh toán (Payment & Billing System):

- **Nạp tiền vào tài khoản:** Hỗ trợ nhiều phương thức nạp tiền phổ biến tại Việt Nam (ví dụ: chuyển khoản ngân hàng, ví điện tử Momo/ZaloPay, thẻ cào điện thoại, cổng thanh toán trực tuyến như VNPay/Onepay). Hệ thống sẽ tích hợp với các cổng thanh toán này thông qua API.

- **Quản lý số dư:** Tự động cập nhật số dư tài khoản người dùng sau mỗi giao dịch nạp tiền hoặc sử dụng dịch vụ. Mọi thay đổi về số dư đều được ghi lại trong lịch sử giao dịch.
- **Tính phí dịch vụ:** Tự động trừ phí từ số dư tài khoản người dùng khi họ sử dụng một dịch vụ giải toán cụ thể, dựa trên mức phí đã định cấu hình cho từng dịch vụ. Trước khi sử dụng dịch vụ, hệ thống sẽ hiển thị rõ ràng chi phí và yêu cầu người dùng xác nhận.
- **Lịch sử giao dịch:** Cung cấp bản ghi chi tiết về tất cả các giao dịch nạp tiền và thanh toán dịch vụ của người dùng. Người dùng có thể lọc và tìm kiếm lịch sử giao dịch theo thời gian, loại giao dịch và trạng thái.
- **Quản lý gói nạp tiền:** Cho phép người dùng lựa chọn các gói nạp tiền với các mức ưu đãi (bonus) khác nhau để khuyến khích nạp tiền số lượng lớn. Ví dụ: nạp 100.000 VNĐ được tặng thêm 10%.
- **Tích hợp thanh toán qua quét mã QR (Không phí trung gian - Ưu tiên):**
  - **Cơ chế tạo mã QR động (VietQR):** Khi người dùng chọn phương thức nạp tiền bằng quét mã QR, hệ thống sẽ tạo một mã QR động theo chuẩn VietQR. Mã QR này sẽ chứa thông tin tài khoản ngân hàng của bạn (chủ tài khoản, số tài khoản, ngân hàng) và một nội dung chuyển khoản duy nhất cho từng giao dịch (ví dụ: `NAPTIEN_[UserID]_[MaGiaoDich]`).
  - **Cơ chế nhận diện giao dịch (Đối soát qua API ngân hàng/Webhook):**
    - **API ngân hàng:** Hệ thống sẽ tích hợp với API của ngân hàng (nếu ngân hàng cung cấp) để truy vấn lịch sử giao dịch định kỳ hoặc nhận webhook thông báo giao dịch đến. Đây là phương pháp hiệu quả nhất để nhận diện giao dịch tự động.
    - **Web scraping/OCR (Phương án dự phòng/thủ công):** Trong trường hợp ngân hàng không cung cấp API, hệ thống có thể sử dụng các kỹ thuật web scraping hoặc OCR để đọc sao kê ngân hàng trực tuyến. Tuy nhiên, phương pháp này phức tạp hơn, kém ổn định và có thể yêu cầu can thiệp thủ công.
    - **Đối soát tự động:** Khi có giao dịch đến tài khoản ngân hàng của bạn, Payment Service sẽ phân tích nội dung chuyển khoản để trích xuất `UserID` và `MaGiaoDich`. Dựa vào thông tin này, hệ thống sẽ đối soát với các giao dịch đang chờ xử lý trong database.
    - **Cập nhật tài khoản khách hàng:** Nếu giao dịch khớp và số tiền chính xác, Payment Service sẽ tự động cập nhật số dư cho `UserID` tương ứng thông qua User Service (sử dụng Message Broker để đảm bảo tính nhất quán và khả năng phục hồi). Nếu có bất kỳ sai lệch nào (sai nội dung, sai số tiền, hoặc không nhận diện được), giao dịch sẽ được đánh dấu để quản trị viên kiểm tra thủ công).
  - **Xử lý trạng thái giao dịch:** Cung cấp trạng thái giao dịch theo thời gian thực cho

người dùng (đang chờ, thành công, thất bại, cần kiểm tra thủ công).

2.1.4. Quản lý nội dung (Content Management):

- **Trang chủ:** Giới thiệu tổng quan về website, các dịch vụ nổi bật, lợi ích và lời kêu gọi hành động. Trang chủ sẽ được thiết kế hấp dẫn, chuyên nghiệp và dễ dàng điều hướng.
- **Trang giới thiệu công ty/đội ngũ:** Thông tin về sứ mệnh, tầm nhìn, đội ngũ phát triển. Điều này giúp tăng cường sự tin tưởng của người dùng đối với dịch vụ.
- **Trang liên hệ:** Cung cấp thông tin liên hệ và biểu mẫu để người dùng gửi câu hỏi, góp ý. Các yêu cầu từ người dùng sẽ được gửi đến email của quản trị viên.
- **Trang FAQ (Câu hỏi thường gặp):** Giải đáp các thắc mắc phổ biến về cách sử dụng dịch vụ, nạp tiền, bảo mật, v.v. Trang FAQ sẽ được cập nhật thường xuyên dựa trên phản hồi của người dùng.
- **Các trang thông tin khác:** Chính sách bảo mật, điều khoản sử dụng. Các trang này sẽ được viết rõ ràng, minh bạch và tuân thủ các quy định của pháp luật.

2.1.5. Quản trị hệ thống (Admin Panel):

- **Quản lý người dùng:** Xem, tìm kiếm, lọc, kích hoạt/vô hiệu hóa tài khoản người dùng, xem chi tiết lịch sử hoạt động của từng người dùng. Quản trị viên có thể cộng/trừ tiền vào tài khoản người dùng trong trường hợp cần thiết.
- **Quản lý dịch vụ:** Thêm, sửa, xóa, kích hoạt/vô hiệu hóa các dịch vụ giải toán, cấu hình mức phí cho từng dịch vụ. Quản trị viên có thể dễ dàng thêm các dịch vụ mới mà không cần can thiệp vào mã nguồn.
- **Quản lý giao dịch:** Xem, tìm kiếm, lọc tất cả các giao dịch nạp tiền và sử dụng dịch vụ, kiểm tra trạng thái và xử lý các giao dịch cần can thiệp thủ công (ví dụ: xác nhận chuyển khoản ngân hàng).
- **Thống kê và báo cáo:** Cung cấp các biểu đồ và báo cáo về doanh thu, số lượng người dùng mới, số lượt sử dụng dịch vụ, dịch vụ phổ biến nhất, v.v. Các báo cáo này giúp quản trị viên theo dõi tình hình kinh doanh và đưa ra các quyết định chiến lược.
- **Quản lý nội dung:** Cập nhật nội dung các trang tĩnh của website thông qua một trình soạn thảo văn bản trực quan.
- **Quản lý gói nạp tiền:** Thêm, sửa, xóa các gói nạp tiền và cấu hình mức bonus.
- **Quản lý quyền truy cập:** Phân quyền cho các tài khoản quản trị viên khác nhau (ví dụ: super admin, content manager, finance manager). Điều này giúp đảm bảo an toàn và phân công công việc hiệu quả.

2.2. Yêu cầu phi chức năng chi tiết

- **Khả năng mở rộng (Scalability):** Hệ thống phải có khả năng mở rộng dễ dàng theo chiều ngang (horizontal scaling) để đáp ứng lượng người dùng và số lượng dịch vụ tăng lên trong tương lai mà không ảnh hưởng đến hiệu suất. Điều này được đảm bảo bởi kiến trúc Microservices và việc sử dụng các công nghệ hỗ trợ containerization như Docker và Kubernetes.
- **Độ tin cậy (Reliability):** Hệ thống phải hoạt động ổn định 24/7, có khả năng chịu lỗi và phục hồi nhanh chóng sau sự cố. Các cơ chế như retry, circuit breaker, sao lưu dữ liệu định kỳ và triển khai trên nhiều máy chủ sẽ được áp dụng để đảm bảo độ tin cậy cao.
- **Bảo mật (Security):** Bảo vệ toàn diện dữ liệu người dùng, thông tin thanh toán và chống lại các cuộc tấn công mạng. Các biện pháp bảo mật cụ thể bao gồm:
  - **Xác thực mạnh mẽ:** Sử dụng JWT cho API authentication, có thể tích hợp xác thực hai yếu tố (2FA) trong tương lai.
  - **Ủy quyền chi tiết:** Sử dụng Role-Based Access Control (RBAC) để phân quyền truy cập cho người dùng và quản trị viên.
  - **Mã hóa dữ liệu:** Sử dụng HTTPS cho tất cả các kết nối, mã hóa mật khẩu bằng bcrypt, và mã hóa các dữ liệu nhạy cảm khác trong cơ sở dữ liệu.
  - **Kiểm tra đầu vào:** Validate tất cả các dữ liệu đầu vào từ người dùng để chống lại các cuộc tấn công như SQL injection, XSS.
  - **Giám sát an ninh:** Sử dụng các công cụ giám sát để phát hiện và cảnh báo về các hoạt động đáng ngờ.
- **Hiệu suất (Performance):** Thời gian phản hồi của website phải nhanh (dưới 2-3 giây cho hầu hết các tác vụ), đảm bảo trải nghiệm người dùng mượt mà ngay cả khi có nhiều người truy cập đồng thời. Các biện pháp tối ưu hóa hiệu suất bao gồm:
  - **Caching:** Sử dụng Redis để cache các dữ liệu thường xuyên được truy cập (ví dụ: thông tin dịch vụ, session người dùng).
  - **Tối ưu hóa truy vấn database:** Sử dụng index, tối ưu hóa các câu lệnh SQL và sử dụng các công cụ giám sát hiệu suất database.
  - **Cân bằng tải:** Sử dụng load balancer để phân phối lưu lượng truy cập đến các máy chủ backend.
- **Dễ bảo trì (Maintainability):** Mã nguồn phải rõ ràng, dễ hiểu, tuân thủ các nguyên tắc thiết kế tốt (SOLID, DRY), dễ sửa đổi và nâng cấp. Kiến trúc Microservices và việc sử dụng các công nghệ hiện đại sẽ hỗ trợ rất nhiều cho yêu cầu này.
- **Khả năng sử dụng (Usability):** Giao diện người dùng phải thân thiện, trực quan, dễ dàng điều hướng và tương thích trên nhiều thiết bị (responsive design cho cả desktop và mobile). Quá trình thiết kế UI/UX sẽ được chú trọng để đảm bảo trải nghiệm người dùng tốt nhất.

- **Khả năng tích hợp (Integrability):** Dễ dàng tích hợp với các hệ thống bên thứ ba (ví dụ: cổng thanh toán, dịch vụ gửi email, SMS) và quan trọng hơn là khả năng tích hợp các dịch vụ toán học mới một cách linh hoạt thông qua API chuẩn hóa. Các API sẽ được thiết kế theo chuẩn RESTful.
- **Khả năng giám sát (Observability):** Hệ thống cần có khả năng thu thập log, metrics và traces để dễ dàng giám sát hiệu suất, phát hiện lỗi và debug. Các công cụ như Prometheus, Grafana, ELK Stack sẽ được sử dụng để xây dựng một hệ thống giám sát toàn diện.

2.3. Kiến trúc Microservices

Để đáp ứng các yêu cầu về khả năng mở rộng, dễ bảo trì và đặc biệt là khả năng tích hợp các dịch vụ "n-..." một cách dễ dàng trong tương lai, **kiến trúc Microservices** là lựa chọn tối ưu. Thay vì xây dựng một ứng dụng nguyên khối (monolithic) lớn và phức tạp, chúng ta sẽ chia nhỏ hệ thống thành các dịch vụ độc lập, nhỏ gọn, có khả năng triển khai và quản lý riêng biệt.

Ưu điểm vượt trội của kiến trúc Microservices cho dự án này:

- **Khả năng mở rộng độc lập:** Mỗi microservice có thể được mở rộng (scale up/out) độc lập dựa trên nhu cầu sử dụng. Ví dụ, nếu dịch vụ giải phương trình bậc 2 có lượng yêu cầu cao, chúng ta có thể tăng số lượng instance của Math Solver Service mà không cần mở rộng toàn bộ hệ thống.
- **Dễ dàng phát triển và triển khai:** Các nhóm nhỏ có thể phát triển và triển khai các dịch vụ riêng biệt mà không ảnh hưởng đến toàn bộ hệ thống. Điều này đặc biệt quan trọng cho việc tích hợp các ứng dụng "n-...": một dịch vụ toán học mới có thể được phát triển và triển khai như một microservice độc lập mà không cần thay đổi các dịch vụ hiện có.
- **Công nghệ đa dạng (Polyglot Persistence & Programming):** Mỗi microservice có thể sử dụng công nghệ (ngôn ngữ lập trình, framework, cơ sở dữ liệu) phù hợp nhất với nghiệp vụ của nó. Ví dụ, Math Solver Service có thể sử dụng Python với các thư viện toán học mạnh mẽ, trong khi Payment Service có thể dùng FastAPI để đảm bảo tính ổn định và bảo mật cao.
- **Khả năng phục hồi (Resilience):** Nếu một microservice gặp sự cố (ví dụ: Math Solver Service bị lỗi), nó sẽ không làm sập toàn bộ hệ thống. Các dịch vụ khác (User, Payment, Content) vẫn có thể hoạt động bình thường, giúp giảm thiểu tác động đến người dùng.
- **Dễ bảo trì và nâng cấp:** Việc sửa đổi hoặc nâng cấp một dịch vụ nhỏ sẽ ít rủi ro và nhanh chóng hơn so với việc thay đổi một ứng dụng nguyên khối lớn. Điều này giúp giảm thời gian downtime và tăng tốc độ phát triển tính năng mới.

Các thành phần chính trong kiến trúc Microservices:

- **Client (Giao diện người dùng - Frontend):** Là ứng dụng web (hoặc mobile app trong tương lai) mà người dùng tương tác trực tiếp. Nó sẽ giao tiếp với các microservice thông qua API Gateway.
- **API Gateway:** Đóng vai trò là điểm truy cập duy nhất cho tất cả các yêu cầu từ client. API Gateway sẽ định tuyến các yêu cầu đến microservice phù hợp, xử lý xác thực, ủy quyền, cân bằng tải, giới hạn tốc độ, và có thể cả caching. Nó giúp ẩn đi sự phức tạp của kiến trúc backend từ phía client.
- **Microservices:** Các dịch vụ độc lập, mỗi dịch vụ chịu trách nhiệm cho một nghiệp vụ cụ thể. Các microservice chính đã được phân tích ở mục 2.4.
- **Database cho từng Microservice:** Mỗi microservice sở hữu cơ sở dữ liệu riêng của mình, đảm bảo tính độc lập và giảm thiểu sự phụ thuộc giữa các dịch vụ. Điều này còn được gọi là **Database per Service** pattern, một nguyên tắc quan trọng trong Microservices.
- **Message Broker (Hệ thống hàng đợi tin nhắn):** Được sử dụng để các microservice giao tiếp bất đồng bộ với nhau. Điều này đặc biệt hữu ích cho các tác vụ không yêu cầu phản hồi ngay lập tức hoặc khi một dịch vụ cần thông báo cho nhiều dịch vụ khác về một sự kiện đã xảy ra (ví dụ: khi một giao dịch thanh toán hoàn tất, Payment Service có thể gửi một thông báo đến Message Broker, và User Service sẽ lắng nghe thông báo này để cập nhật số dư).
- **Service Discovery:** Cơ chế để các microservice tìm thấy nhau và giao tiếp. Khi một microservice mới được triển khai, nó sẽ đăng ký với Service Discovery. Các microservice khác hoặc API Gateway có thể truy vấn Service Discovery để tìm địa chỉ của microservice cần gọi.
- **Logging & Monitoring (Ghi log và Giám sát):** Hệ thống tập trung để thu thập log từ tất cả các microservice, giúp theo dõi hoạt động, phát hiện lỗi và debug. Đồng thời, các công cụ giám sát sẽ thu thập các chỉ số hiệu suất (metrics) của từng microservice và toàn bộ hệ thống, cung cấp cái nhìn tổng quan về tình trạng sức khỏe của ứng dụng.

2.4. Phân tích các Microservice chính

Dựa trên yêu cầu chức năng và nguyên tắc của kiến trúc Microservices, chúng ta có thể phân chia hệ thống thành các microservice chính sau. Mỗi microservice sẽ là một đơn vị phát triển, triển khai và mở rộng độc lập:

- **User Service (Dịch vụ Người dùng):**
    - **Chức năng chính:** 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. Nó cũng sẽ lưu trữ lịch sử sử dụng dịch vụ của người dùng ở mức

tổng quan (ví dụ: chỉ lưu ID dịch vụ và thời gian sử dụng, không lưu chi tiết dữ liệu đầu vào/đầu ra của bài toán).

- **Giao tiếp:** Cung cấp API cho Frontend và các microservice khác (ví dụ: Math Solver Service để kiểm tra/trừ tiền, Payment Service để cộng tiền).
- **Database:** Sở hữu cơ sở dữ liệu PostgreSQL riêng để lưu trữ thông tin người dùng và số dư.

- **Payment Service (Dịch vụ Thanh toán):**
  - **Chức năng chính:** 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 (ví dụ: VNPay, Momo, ngân hàng). Nó sẽ quản lý lịch sử các giao dịch nạp tiền và thanh toán dịch vụ.
  - **Giao tiếp:** Cung cấp API cho Frontend để khởi tạo giao dịch nạp tiền. Giao tiếp với User Service (thông qua Message Broker hoặc API trực tiếp) để cập nhật số dư sau khi giao dịch thành công. Nhận webhook từ các cổng thanh toán bên ngoài.
  - **Database:** Sở hữu cơ sở dữ liệu PostgreSQL riêng để lưu trữ thông tin giao dịch thanh toán.
- **Math Solver Service (Dịch vụ Giải toán):**
  - **Chức năng chính:** Đây là dịch vụ lõi chứa logic giải các bài toán toán học (ví dụ: giải phương trình bậc 2, giải hệ phương trình). Nó sẽ cung cấp các API để Frontend hoặc các ứng dụng khác có thể gọi đến để sử dụng các chức năng giải toán.
  - **Khả năng mở rộng:** Mỗi loại bài toán có thể là một module nhỏ hơn bên trong dịch vụ này (nếu logic đơn giản) hoặc thậm chí là một microservice riêng biệt (nếu logic phức tạp). Điều này cho phép tích hợp dễ dàng các dịch vụ toán học mới trong tương lai.
  - **Giao tiếp:** Cung cấp API cho Frontend để nhận yêu cầu giải toán. Giao tiếp với User Service để kiểm tra số dư và trừ phí.
  - **Database:** Sở hữu cơ sở dữ liệu PostgreSQL riêng để lưu trữ lịch sử giải toán chi tiết (dữ liệu đầu vào, đầu ra, thời gian xử lý).
- **Content Service (Dịch vụ Nội dung):**
  - **Chức năng chính:** 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. Cung cấp API để Frontend có thể lấy nội dung và cho Admin Panel có thể cập nhật nội dung.
  - **Giao tiếp:** Cung cấp API cho Frontend để lấy nội dung các trang. Cung cấp API cho Admin Panel để quản lý nội dung.
  - **Database:** Sở hữu cơ sở dữ liệu PostgreSQL riêng để lưu trữ nội dung website.

- **Admin Service (Dịch vụ Quản trị):**
  - **Chức năng chính:** 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.
  - **Giao tiếp:** Cung cấp API cho Admin Panel (Frontend). Giao tiếp với tất cả các microservice khác để lấy dữ liệu và thực hiện các thao tác quản trị.
  - **Database:** Có thể có cơ sở dữ liệu PostgreSQL riêng để lưu trữ cấu hình hệ thống, log audit, hoặc có thể không cần database riêng nếu chỉ đóng vai trò aggregator.

2.5. Chiến lược tích hợp dịch vụ toán học mới

Một trong những yêu cầu quan trọng nhất của dự án là khả năng tích hợp dễ dàng các dịch vụ toán học mới (các phần "n-...") trong tương lai. Với kiến trúc Microservices, việc này được thực hiện theo các cách sau:

2.5.1. Microservice độc lập cho từng dịch vụ toán học:
Mỗi dịch vụ toán học mới (ví dụ: Giải phương trình bậc nhất, Tích phân, Đạo hàm, v.v.) sẽ được phát triển như một microservice hoàn toàn độc lập với:

- **Database riêng:** Mỗi dịch vụ có cơ sở dữ liệu PostgreSQL riêng để lưu trữ lịch sử giải toán và cấu hình riêng của nó.
- **API chuẩn hóa:** Tất cả các dịch vụ toán học đều tuân thủ một chuẩn API chung để Frontend có thể gọi đến một cách nhất quán.
- **Công nghệ linh hoạt:** Mỗi dịch vụ có thể sử dụng ngôn ngữ lập trình và thư viện phù hợp nhất (ví dụ: Python với NumPy/SciPy cho toán học, R cho thống kê, v.v.).

2.5.2. Quy trình tích hợp:

1. **Phát triển độc lập:** Dịch vụ toán học mới được phát triển như một microservice riêng biệt.
2. **Đăng ký với Service Discovery:** Sau khi hoàn thành, dịch vụ sẽ đăng ký với hệ thống Service Discovery.
3. **Cập nhật API Gateway:** Cấu hình API Gateway để định tuyến các yêu cầu đến dịch vụ mới.
4. **Cập nhật Frontend:** Thêm giao diện cho dịch vụ mới trong Frontend.
5. **Cập nhật Admin Panel:** Thêm khả năng quản lý dịch vụ mới trong Admin Panel.

2.5.3. Ví dụ cụ thể - Tích hợp "Giải phương trình bậc nhất":
Giả sử chúng ta muốn thêm dịch vụ "Giải phương trình bậc nhất":

1. Tạo một microservice mới `Linear Equation Solver Service` với API endpoint `/api/linear-equation/solve`.
2. Service này có database riêng để lưu lịch sử giải phương trình bậc nhất.
3. Cập nhật API Gateway để định tuyến `/api/linear-equation/*` đến service mới.
4. Frontend thêm trang mới cho phép người dùng nhập phương trình bậc nhất và hiển thị kết quả.
5. Admin Panel thêm khả năng quản lý dịch vụ "Giải phương trình bậc nhất" (cấu hình phí, kích hoạt/vô hiệu hóa).

Quá trình này hoàn toàn không ảnh hưởng đến các dịch vụ hiện có và có thể được thực hiện song song với việc phát triển các tính năng khác.

3. Đề xuất công nghệ và công cụ

Dựa trên phân tích dự án Suna và yêu cầu của dự án, chúng ta sẽ sử dụng stack công nghệ hiện đại và đã được kiểm chứng:

3.1. Frontend

3.1.1. Next.js (React Framework)

Next.js là framework React full-stack mạnh mẽ, cung cấp nhiều tính năng tối ưu hóa hiệu suất và trải nghiệm phát triển:

- **Server-Side Rendering (SSR) và Static Site Generation (SSG):** Cải thiện SEO và tốc độ tải trang ban đầu.
- **API Routes:** Cho phép xây dựng API endpoints ngay trong ứng dụng Next.js, hữu ích cho việc xử lý authentication và proxy requests.
- **Automatic Code Splitting:** Tự động chia nhỏ code để tối ưu hóa tốc độ tải.
- **Built-in CSS Support:** Hỗ trợ CSS Modules, Sass và CSS-in-JS.
- **Image Optimization:** Tối ưu hóa hình ảnh tự động.

3.1.2. TypeScript

TypeScript mang lại type safety và cải thiện trải nghiệm phát triển:

- **Type Safety:** Giảm thiểu lỗi runtime thông qua kiểm tra type tại compile time.
- **Better IDE Support:** IntelliSense, auto-completion và refactoring tốt hơn.
- **Maintainability:** Code dễ đọc, dễ hiểu và dễ bảo trì hơn.

3.1.3. Tailwind CSS

Tailwind CSS là utility-first CSS framework:

- **Rapid Development:** Phát triển UI nhanh chóng với các utility classes.
- **Consistent Design:** Đảm bảo tính nhất quán trong thiết kế.
- **Responsive Design:** Dễ dàng tạo responsive design.
- **Customizable:** Có thể tùy chỉnh theo design system riêng.

3.1.4. Các thư viện hỗ trợ:

- **React Query (TanStack Query):** Quản lý state server và caching API calls hiệu quả.
- **React Hook Form:** Quản lý form với hiệu suất cao và validation mạnh mẽ.
- **Zustand hoặc Redux Toolkit:** Quản lý global state.
- **Framer Motion:** Animation và transitions mượt mà.

3.2. Backend

3.2.1. FastAPI (Python)

FastAPI là framework Python hiện đại cho việc xây dựng API:

- **High Performance:** Hiệu suất cao, tương đương với NodeJS và Go.
- **Type Hints:** Sử dụng Python type hints để tự động validation và serialization.
- **Automatic API Documentation:** Tự động tạo OpenAPI (Swagger) documentation.
- **Async Support:** Hỗ trợ async/await natively.
- **Dễ học và sử dụng:** Cú pháp Python đơn giản, dễ đọc, giúp tăng tốc độ phát triển.
- **Hệ sinh thái phong phú:** Python có hệ sinh thái thư viện khổng lồ, đặc biệt là cho các tác vụ tính toán khoa học và xử lý dữ liệu, rất phù hợp cho Math Solver Service.

3.2.2. Cơ sở dữ liệu: PostgreSQL

PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở mạnh mẽ, ổn định và đáng tin cậy. Nó sẽ được sử dụng làm cơ sở dữ liệu chính cho các microservice, đảm bảo tính toàn vẹn dữ liệu và khả năng mở rộng. PostgreSQL hỗ trợ nhiều tính năng nâng cao như JSONB, indexing, và khả năng mở rộng thông qua các extension, rất phù hợp cho các nghiệp vụ đa dạng của dự án.

3.2.3. Hàng đợi tin nhắn/Tác vụ nền: Dramatiq

Dramatiq là một thư viện Python nhẹ để xử lý các tác vụ nền (background tasks) và hàng đợi tin nhắn. Nó sẽ được sử dụng cho các tác vụ không đồng bộ như gửi email thông báo, xử lý các yêu cầu giải toán phức tạp cần nhiều thời gian, hoặc cập nhật dữ liệu sau khi giao dịch thanh toán hoàn tất.

3.2.4. Caching: Redis

Redis sẽ được sử dụng làm cache layer để:

- Cache kết quả API calls thường xuyên.

- Lưu trữ session data.
- Cache kết quả tính toán phức tạp.
- Làm message broker cho Dramatiq.

3.2.5. ORM: SQLAlchemy

SQLAlchemy là ORM mạnh mẽ cho Python:

- **Flexibility:** Hỗ trợ cả ORM pattern và Core (raw SQL).
- **Database Agnostic:** Hỗ trợ nhiều loại database.
- **Migration Support:** Alembic cho database migrations.
- **Performance:** Tối ưu hóa queries và lazy loading.

3.3. DevOps và Infrastructure

3.3.1. Containerization: Docker

- **Consistency:** Đảm bảo môi trường nhất quán giữa development, staging và production.
- **Scalability:** Dễ dàng scale các microservice độc lập.
- **Deployment:** Đơn giản hóa quá trình deployment.

3.3.2. Orchestration: Docker Compose (Development) / Kubernetes (Production)

- **Docker Compose:** Cho môi trường development và testing.
- **Kubernetes:** Cho production environment với auto-scaling, service discovery, load balancing.

3.3.3. API Gateway: Traefik hoặc Kong

- **Load Balancing:** Phân phối tải giữa các instance của microservice.
- **SSL Termination:** Xử lý HTTPS certificates.
- **Rate Limiting:** Giới hạn số lượng requests.
- **Authentication:** Centralized authentication handling.

3.3.4. Monitoring và Logging:

- **Prometheus + Grafana:** Metrics collection và visualization.
- **ELK Stack (Elasticsearch, Logstash, Kibana):** Centralized logging.
- **Sentry:** Error tracking và performance monitoring.

3.4. Bảo mật

3.4.1. Authentication & Authorization:

- **JWT (JSON Web Tokens):** Cho API authentication.
- **OAuth 2.0:** Cho third-party integrations.
- **RBAC (Role-Based Access Control):** Phân quyền chi tiết.

3.4.2. Data Protection:

- **HTTPS/TLS:** Mã hóa data in transit.
- **bcrypt:** Hash passwords.
- **Database encryption:** Mã hóa sensitive data trong database.

3.4.3. Security Headers:

- **CORS:** Cross-Origin Resource Sharing configuration.
- **CSP:** Content Security Policy.
- **HSTS:** HTTP Strict Transport Security.

3.5. Payment Integration

3.5.1. Cổng thanh toán Việt Nam:

- **VNPay:** Cổng thanh toán phổ biến tại Việt Nam.
- **Momo/ZaloPay:** Ví điện tử.
- **OnePay:** Thanh toán thẻ quốc tế.

3.5.2. Webhook Handling:

- Xử lý webhook từ các cổng thanh toán để cập nhật trạng thái giao dịch real-time.

3.6. Lý do lựa chọn công nghệ

Việc lựa chọn công nghệ dựa trên dự án Suna mang lại nhiều lợi ích cho dự án website dịch vụ toán học:

- **Tính hiện đại và hiệu suất:** Next.js và FastAPI là các công nghệ hiện đại, cung cấp hiệu suất cao và trải nghiệm phát triển tốt.
- **Khả năng mở rộng:** Kiến trúc Microservices và việc sử dụng Docker cho phép mở rộng linh hoạt từng thành phần của hệ thống.
- **Tốc độ phát triển:** Việc sử dụng các thư viện như React Query, React Hook Form, Tailwind CSS giúp tăng tốc độ phát triển frontend.
- **Hệ sinh thái phong phú:** Python có hệ sinh thái mạnh mẽ cho toán học và khoa học dữ liệu, rất phù hợp cho các dịch vụ giải toán. React/Next.js có cộng đồng lớn và nhiều tài nguyên hỗ trợ.

- **Bảo mật:** Các công nghệ như JWT, HTTPS, bcrypt cung cấp các giải pháp bảo mật cần thiết.
- **Chi phí hiệu quả:** Sử dụng các công nghệ mã nguồn mở giúp giảm chi phí licensing.
- **Khả năng tuyển dụng:** Các công nghệ phổ biến giúp dễ dàng tìm kiếm và tuyển dụng nhân tài.

SStack công nghệ này đảm bảo dự án có thể phát triển nhanh chóng, dễ dàng bảo trì và mở rộng trong tương lai, đồng thời đáp ứng tất cả các yêu cầu về hiệu suất, bảo mật và khả năng tích hợp.g).

2.1.3.1. Tích hợp thanh toán qua quét mã QR (Không phí trung gian - Ưu tiên):

- **Cơ chế tạo mã QR động (VietQR):** Khi người dùng chọn phương thức nạp tiền bằng quét mã QR, hệ thống sẽ tạo một mã QR động theo chuẩn VietQR. Mã QR này sẽ chứa thông tin tài khoản ngân hàng của bạn (chủ tài khoản, số tài khoản, ngân hàng) và một nội dung chuyển khoản duy nhất cho từng giao dịch (ví dụ: `NAPTIEN_[UserID]_[MaGiaoDich]`).
- **Cơ chế nhận diện giao dịch (Đối soát qua API ngân hàng/Webhook):**
  - **API ngân hàng:** Hệ thống sẽ tích hợp với API của ngân hàng (nếu ngân hàng cung cấp) để truy vấn lịch sử giao dịch định kỳ hoặc nhận webhook thông báo giao dịch đến. Đây là phương pháp hiệu quả nhất để nhận diện giao dịch tự động.
  - **Web scraping/OCR (Phương án dự phòng/thủ công):** Trong trường hợp ngân hàng không cung cấp API, hệ thống có thể sử dụng các kỹ thuật web scraping hoặc OCR để đọc sao kê ngân hàng trực tuyến. Tuy nhiên, phương pháp này phức tạp hơn, kém ổn định và có thể yêu cầu can thiệp thủ công.
  - **Đối soát tự động:** Khi có giao dịch đến tài khoản ngân hàng của bạn, Payment Service sẽ phân tích nội dung chuyển khoản để trích xuất `UserID` và `MaGiaoDich`. Dựa vào thông tin này, hệ thống sẽ đối soát với các giao dịch đang chờ xử lý trong database.
  - **Cập nhật tài khoản khách hàng:** Nếu giao dịch khớp và số tiền chính xác, Payment Service sẽ tự động cập nhật số dư cho `UserID` tương ứng thông qua User Service (sử dụng Message Broker để đảm bảo tính nhất quán và khả năng phục hồi). Nếu có bất kỳ sai lệch nào (sai nội dung, sai số tiền, hoặc không nhận diện được), giao dịch sẽ được đánh dấu để quản trị viên kiểm tra thủ công.
- **Xử lý trạng thái giao dịch:** Cung cấp trạng thái giao dịch theo thời gian thực cho người dùng (đang chờ, thành công, thất bại, cần kiểm tra thủ công).