Cấu trúc thư mục chi tiết cho từng Microservice

Dưới đây là cấu trúc thư mục đề xuất cho từng microservice trong dự án Website Dịch vụ Toán học, bao gồm cả phần liên quan đến database. Mỗi thư mục và file đều có vai trò cụ thể trong việc tổ chức mã nguồn và triển khai dự án.

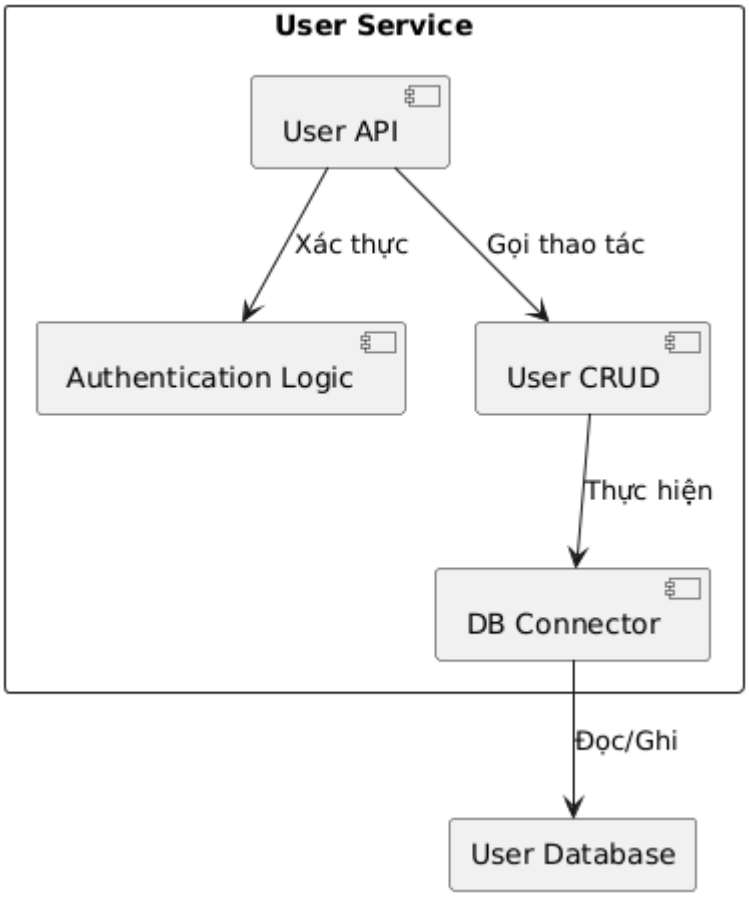
1. User Service

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.

Plain Text

```
user-service/
├── app/                          # Thư mục chứa mã nguồn chính của ứng dụng
FastAPI
│   ├── api/                      # Định nghĩa các API endpoints và routers
│   │   ├── __init__.py           # File khởi tạo package Python
│   │   └── v1/                   # Phiên bản API (ví dụ: v1), giúp quản lý
phiên bản API dễ dàng
│   │       ├── __init__.py       # File khởi tạo package Python
│   │       ├── endpoints/        # Chứa các file định nghĩa endpoint cụ thể
(ví dụ: users.py, auth.py)
│   │       │   ├── __init__.py
│   │       │   ├── users.py      # API cho các thao tác với người dùng (GET,
PUT, DELETE user)
│   │       │   └── auth.py       # API cho đăng ký, đăng nhập, refresh token
│   │       └── deps.py           # Chứa các hàm dependency injection (ví dụ:
get_current_user để xác thực)
│   ├── core/                     # Cấu hình chung, hằng số, các tiện ích cốt
lõi
│   │   ├── __init__.py           # File khởi tạo package Python
│   │   ├── config.py             # Chứa các biến môi trường, cấu hình ứng
dụng
│   │   └── security.py           # Xử lý JWT (JSON Web Token), mã hóa mật
khẩu (bcrypt)
│   ├── crud/                     # Create, Read, Update, Delete - Các thao
tác tương tác trực tiếp với database
│   │   ├── __init__.py           # File khởi tạo package Python
│   │   └── user.py               # Các hàm CRUD cho model User (ví dụ:
create_user, get_user_by_email)
│   ├── db/                       # Cấu hình database và định nghĩa các model
│   │   ├── __init__.py           # File khởi tạo package Python
│   │   ├── base.py               # Base class cho các model SQLAlchemy, định
```

```
nghĩa các cột chung
│   │   ├── session.py           # Hàm khởi tạo session database, quản lý
kết nối DB
│   │   └── models/              # Định nghĩa các model SQLAlchemy ánh xạ
tới bảng trong database
│   │       ├── __init__.py      # File khởi tạo package Python
│   │       └── user.py          # Định nghĩa model User (bao gồm thông tin
cá nhân, số dư)
│   ├── schemas/                 # Định nghĩa các Pydantic schemas cho
validation dữ liệu (request/response)
│   │   ├── __init__.py          # File khởi tạo package Python
│   │   ├── user.py              # Schema cho User (ví dụ: UserCreate,
UserResponse)
│   │   └── token.py             # Schema cho JWT token (AccessToken,
RefreshToken)
│   └── main.py                  # File khởi tạo ứng dụng FastAPI chính
├── alembic/                     # Thư mục chứa các script migration
database (do Alembic tạo)
│   ├── versions/                # Chứa các file migration cụ thể (ví dụ:
123abc_initial_migration.py)
│   └── env.py                   # Cấu hình môi trường cho Alembic
├── tests/                       # Thư mục chứa các bài kiểm thử (unit
tests, integration tests)
│   ├── __init__.py
│   ├── api/                     # Tests cho các API endpoints
│   └── crud/                    # Tests cho các hàm CRUD
├── Dockerfile                   # Định nghĩa Docker image cho User Service
├── requirements.txt             # Danh sách các thư viện Python cần thiết
cho dự án
├── .env.example                 # File mẫu chứa các biến môi trường cần
thiết
├── README.md                    # File mô tả dự án, hướng dẫn cài đặt và sử
dụng
└── alembic.ini                  # File cấu hình cho Alembic
```



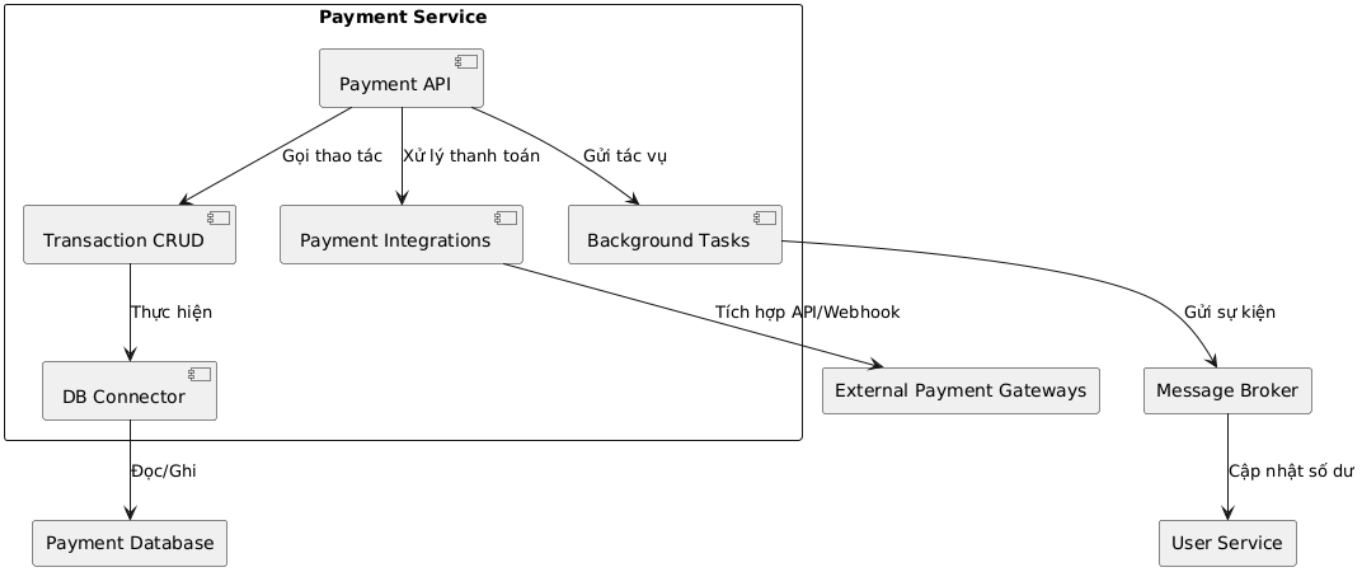
2. Payment Service

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.

Plain Text

```
payment-service/
├── app/
│   ├── api/
│   │   ├── __init__.py
│   │   └── v1/
│   │       ├── __init__.py
│   │       ├── endpoints/   # payment.py (API nạp tiền), webhook.py (nhận
thông báo từ cổng TT)
│   │       └── deps.py
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── security.py
│   ├── crud/
│   │   ├── __init__.py
│   │   ├── transaction.py   # CRUD cho giao dịch nạp/thanh toán
```

```
│   │   └── payment_package.py # CRUD cho các gói nạp tiền
│   ├── db/
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── session.py
│   │   └── models/      # transaction.py, payment_package.py
│   │       ├── __init__.py
│   │       └── transaction.py # Định nghĩa model Transaction
│   ├── integrations/    # Logic tích hợp với các cổng thanh toán bên ngoài
│   │   ├── __init__.py
│   │   ├── vnpay.py     # Logic tích hợp VNPay
│   │   ├── momo.py      # Logic tích hợp Momo
│   │   └── bank_api.py  # Logic tích hợp API ngân hàng (nếu có)
│   ├── schemas/
│   │   ├── __init__.py
│   │   ├── payment.py   # Schema cho yêu cầu nạp tiền
│   │   └── transaction.py # Schema cho thông tin giao dịch
│   ├── tasks/           # Các tác vụ chạy nền (Dramatiq) để xử lý bất đồng
bộ
│   │   ├── __init__.py
│   │   └── update_user_balance.py # Tác vụ cập nhật số dư người dùng sau
khi thanh toán thành công
│   └── main.py
├── alembic/
├── tests/
├── Dockerfile
├── requirements.txt
├── .env.example
├── README.md
└── alembic.ini
```

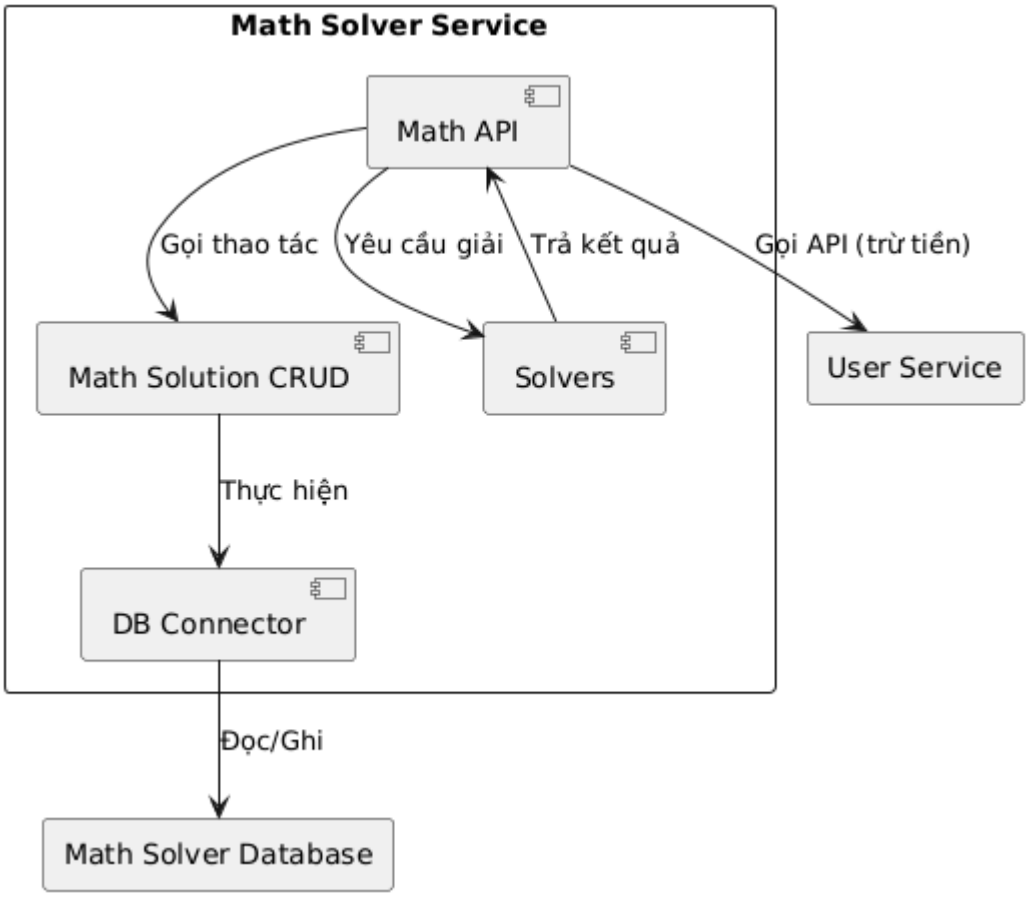


3. Math Solver Service

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.

Plain Text

```
math-solver-service/
├── app/
│   ├── api/
│   │   ├── __init__.py
│   │   └── v1/
│   │       ├── __init__.py
│   │       ├── endpoints/   # quadratic_equation.py, system_of_equations.py
│   │       └── deps.py
│   ├── core/
│   │   ├── __init__.py
│   │   └── config.py
│   ├── crud/
│   │   ├── __init__.py
│   │   └── math_solution.py # CRUD cho lịch sử giải toán
│   ├── db/
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── session.py
│   │   └── models/      # math_solution.py
│   │       ├── __init__.py
│   │       └── math_solution.py # Định nghĩa model MathSolution (lưu trữ
input/output bài toán)
│   ├── schemas/
│   │   ├── __init__.py
│   │   ├── math.py      # Schema cho input bài toán
│   │   └── solution.py  # Schema cho output bài toán
│   ├── solvers/         # Chứa logic giải các bài toán toán học cụ thể
│   │   ├── __init__.py
│   │   ├── quadratic.py     # Logic giải phương trình bậc 2
│   │   └── linear_system.py # Logic giải hệ phương trình
│   └── main.py
├── alembic/
├── tests/
├── Dockerfile
├── requirements.txt
├── .env.example
├── README.md
└── alembic.ini
```



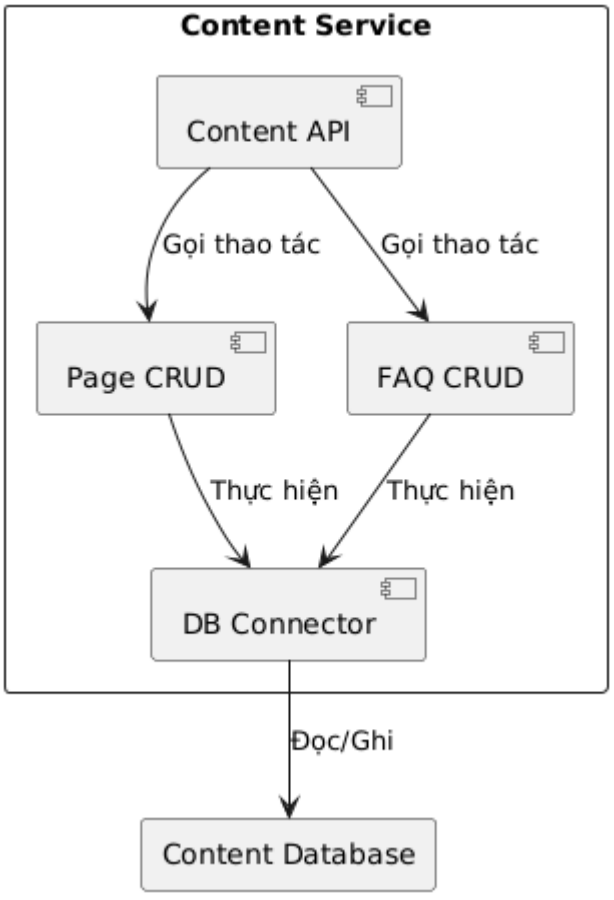
4. Content Service

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.

Plain Text

```
content-service/
├── app/
│   ├── api/
│   │   ├── __init__.py
│   │   └── v1/
│   │       ├── __init__.py
│   │       ├── endpoints/   # pages.py, faqs.py
│   │       └── deps.py
│   ├── core/
│   │   ├── __init__.py
│   │   └── config.py
│   ├── crud/
│   │   ├── __init__.py
│   │   ├── page.py        # CRUD cho các trang tĩnh
│   │   └── faq.py         # CRUD cho các câu hỏi thường gặp
│   ├── db/
```

```
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── session.py
│   │   └── models/      # page.py, faq.py
│   │       ├── __init__.py
│   │       └── page.py    # Định nghĩa model Page
│   ├── schemas/
│   │   ├── __init__.py
│   │   ├── page.py
│   │   └── faq.py
│   └── main.py
├── alembic/
├── tests/
├── Dockerfile
├── requirements.txt
├── .env.example
├── README.md
└── alembic.ini
```



5. Admin Service

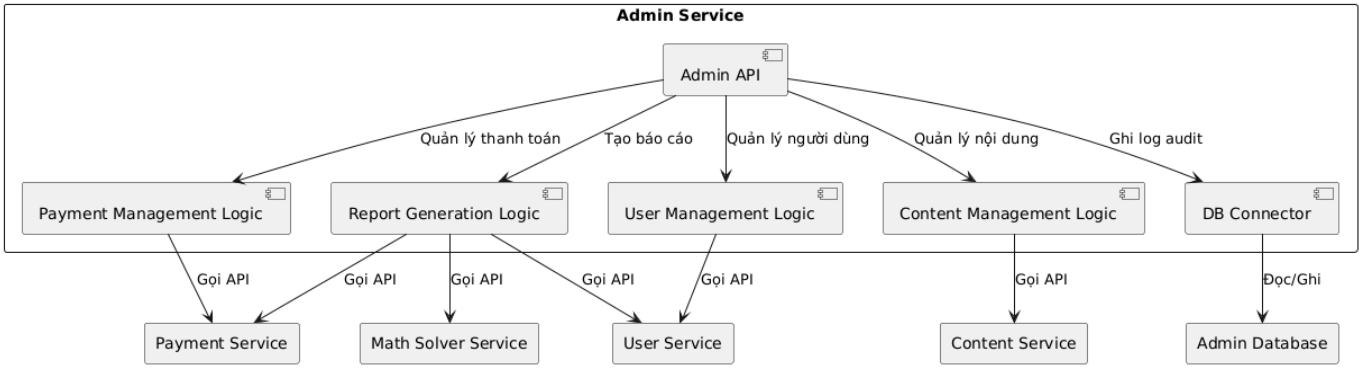
Mục đích: Cung cấp các chức năng quản trị hệ thống: quản lý người dùng, quản lý dịch vụ, quản lý giao dịch, thống kê báo cáo, quản lý quyền truy cập. Đây là dịch vụ tổng hợp, gọi

đến các microservice khác để lấy dữ liệu và thực hiện các thao tác quản trị.

Plain Text

```
admin-service/
├── app/
│   ├── api/
│   │   ├── __init__.py
│   │   └── v1/
│   │       ├── __init__.py
│   │       ├── endpoints/   # users.py, services.py, transactions.py,
reports.py, content.py, auth.py
│   │       └── deps.py
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── security.py
│   ├── crud/            # Các thao tác CRUD (có thể gọi đến các service
khác hoặc DB riêng nếu có)
│   │   ├── __init__.py
│   │   └── admin_user.py # CRUD cho tài khoản admin (nếu có DB riêng cho
admin)
│   ├── db/              # Cấu hình database và model cho Admin Service (nếu
có DB riêng)
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── session.py
│   │   └── models/      # admin_user.py, audit_log.py
│   │       ├── __init__.py
│   │       └── admin_user.py # Định nghĩa model AdminUser
│   ├── schemas/
│   │   ├── __init__.py
│   │   ├── admin.py     # Schema cho các đối tượng quản trị
│   │   └── report.py    # Schema cho báo cáo
│   ├── services/        # Chứa logic gọi đến các microservice khác để thực
hiện các tác vụ quản trị
│   │   ├── __init__.py
│   │   ├── user_api.py    # Hàm gọi API đến User Service
│   │   └── payment_api.py # Hàm gọi API đến Payment Service
│   └── main.py
├── alembic/             # Migrations database (nếu có DB riêng cho Admin
Service)
├── tests/
├── Dockerfile
├── requirements.txt
├── .env.example
```

```
├── README.md
└── alembic.ini
```



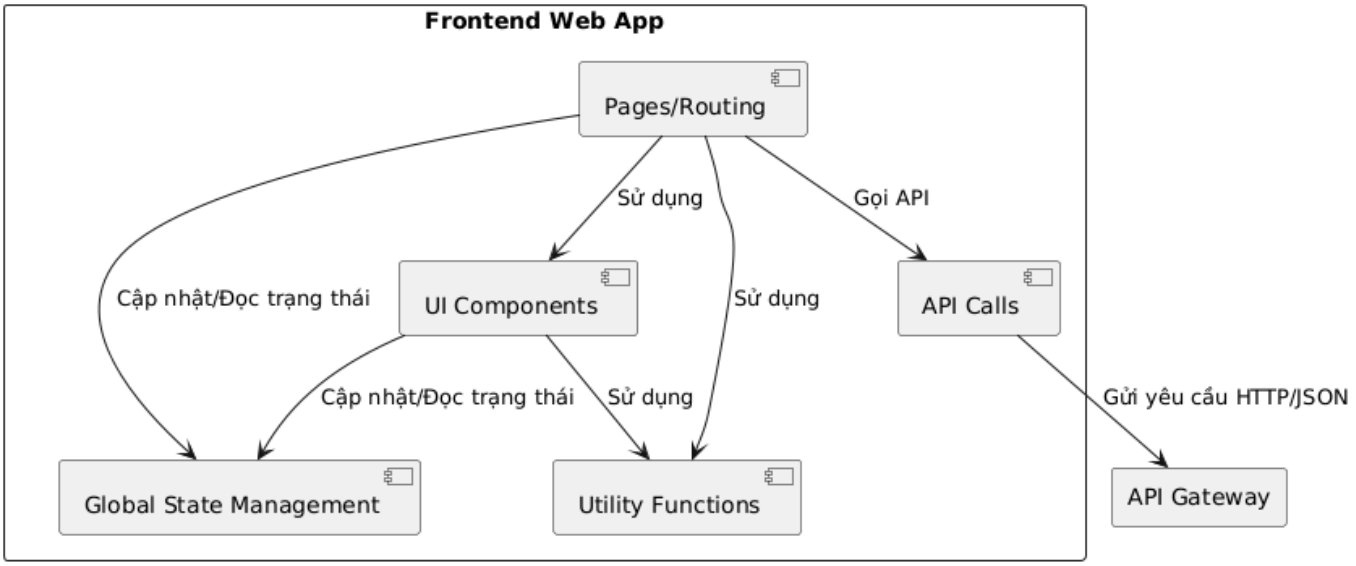
6. Frontend (Web App)

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.

Plain Text

```
frontend-web-app/
├── public/              # Thư mục chứa các file tĩnh (favicon, images) sẽ
được phục vụ trực tiếp
├── src/                 # Thư mục chứa mã nguồn chính của ứng dụng Next.js
│   ├── api/             # Các hàm gọi API đến Backend (sử dụng React Query
để quản lý state và caching)
│   │   ├── __init__.ts
│   │   ├── auth.ts      # Hàm gọi API liên quan đến xác thực
│   │   ├── user.ts      # Hàm gọi API liên quan đến người dùng
│   │   └── math.ts      # Hàm gọi API liên quan đến dịch vụ toán học
│   ├── components/      # Các thành phần UI tái sử dụng (React Components)
│   │   ├── __init__.tsx
│   │   ├── common/      # Các component chung (buttons, inputs, modals)
│   │   ├── layout/      # Các component bố cục trang (header, footer,
sidebar)
│   │   └── ui/          # Các component UI phức tạp hơn
│   ├── hooks/           # Custom React Hooks để tái sử dụng logic trạng thái
│   │   ├── __init__.ts
│   │   └── useAuth.ts   # Hook quản lý trạng thái xác thực người dùng
│   ├── pages/           # Các trang Next.js, định nghĩa routing của ứng dụng
│   │   ├── _app.tsx     # File khởi tạo ứng dụng Next.js, cấu hình global
│   │   ├── _document.tsx # File tùy chỉnh HTML document (cho SSR)
│   │   ├── index.tsx    # Trang chủ
│   │   ├── auth/        # Thư mục chứa các trang liên quan đến xác thực
(đăng nhập, đăng ký, quên mật khẩu)
│   │   ├── user/        # Thư mục chứa các trang quản lý hồ sơ, lịch sử
```

```
giao dịch, lịch sử sử dụng dịch vụ
│   │   ├── math/        # Thư mục chứa các trang dịch vụ toán học (giải
phương trình, hệ phương trình)
│   │   └── admin/       # Thư mục chứa các trang Admin Panel (nếu Admin
Panel là một phần của ứng dụng này)
│   ├── styles/          # Thư mục chứa các file CSS, cấu hình Tailwind CSS
│   │   ├── globals.css  # Global CSS styles
│   │   └── tailwind.config.js # Cấu hình Tailwind CSS
│   ├── types/           # Định nghĩa TypeScript types/interfaces cho dữ liệu
│   │   ├── __init__.ts
│   │   └── api.ts       # Định nghĩa các kiểu dữ liệu cho API
request/response
│   ├── utils/           # Các hàm tiện ích chung, không liên quan đến UI
│   │   ├── __init__.ts
│   │   └── helpers.ts   # Các hàm helper chung
│   └── store/           # Quản lý global state (sử dụng Zustand hoặc Redux
Toolkit)
│       ├── __init__.ts
│       └── authStore.ts # Store quản lý trạng thái xác thực
├── .env.local.example   # File mẫu chứa các biến môi trường cục bộ
├── next.config.js       # Cấu hình Next.js
├── package.json         # Danh sách các dependencies và scripts của dự án
Node.js
├── tsconfig.json        # Cấu hình TypeScript
├── README.md
└── yarn.lock (hoặc package-lock.json) # File khóa dependencies
```



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. Nếu Admin Panel được phát triển như một ứng dụng Frontend riêng biệt (thay vì là một

phần của `frontend-web-app`), cấu trúc thư mục của nó sẽ tương tự như `frontend-web-app`, nhưng tập trung vào các trang và thành phần UI dành cho quản trị viên.

Plain Text

```
admin-panel-app/
├── public/
├── src/
│   ├── api/              # Các hàm gọi API đến Admin Service
│   ├── components/       # Các thành phần UI cho Admin
│   ├── hooks/
│   ├── pages/            # Các trang Admin (dashboard, user management, etc.)
│   ├── styles/
│   ├── types/
│   ├── utils/
│   └── store/
├── .env.local.example
├── next.config.js
├── package.json
├── tsconfig.json
├── README.md
└── yarn.lock (hoặc package-lock.json)
```

Cấu trúc thư mục tổng thể toàn dự án

Đây là cấu trúc thư mục tổng thể của toàn bộ dự án Website Dịch vụ Toán học, tổ chức theo kiến trúc Microservices và bao gồm các thành phần Frontend, Backend, DevOps, và các tài liệu liên quan. Cấu trúc này giúp quản lý dự án lớn một cách có tổ chức, dễ dàng mở rộng và bảo trì.

Plain Text

```
math-service-platform/
├── docs/                       # Thư mục chứa tất cả tài liệu liên quan
đến dự án
│   ├── design/                 # Tài liệu thiết kế (UI/UX, API specs, ERD)
│   ├── architecture/           # Sơ đồ kiến trúc tổng thể và chi tiết
│   ├── requirements/           # Tài liệu yêu cầu chức năng/phi chức năng
│   └── processes/              # Quy trình triển khai, vận hành, bảo trì
├── frontend/                   # Thư mục chứa mã nguồn của tất cả các ứng
dụng Frontend
│   └── web-app/                # Ứng dụng web chính (Next.js) cho người
dùng cuối
│       ├── public/
```

```
│       ├── src/
│       ├── .env.local.example
│       ├── next.config.js
│       ├── package.json
│       └── tsconfig.json
│   └── admin-panel/            # (Tùy chọn) Ứng dụng Admin Panel riêng biệt
│       ├── public/
│       ├── src/
│       ├── .env.local.example
│       ├── next.config.js
│       ├── package.json
│       └── tsconfig.json
├── backend/                    # Thư mục chứa mã nguồn của tất cả các
Microservice Backend
│   ├── user-service/           # User Service (FastAPI) - Quản lý người
dùng
│   │   ├── app/
│   │   ├── alembic/
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   └── alembic.ini
│   ├── payment-service/        # Payment Service (FastAPI) - Xử lý thanh
toán
│   │   ├── app/
│   │   ├── alembic/
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   └── alembic.ini
│   ├── math-solver-service/    # Math Solver Service (FastAPI) - Giải toán
│   │   ├── app/
│   │   ├── alembic/
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   └── alembic.ini
│   ├── content-service/        # Content Service (FastAPI) - Quản lý nội
dung
│   │   ├── app/
│   │   ├── alembic/
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   └── alembic.ini
│   ├── admin-service/          # Admin Service (FastAPI) - Quản lý hệ thống
│   │   ├── app/
│   │   ├── alembic/
```

```
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   └── alembic.ini
│   └── api-gateway/             # Cấu hình API Gateway (Traefik/Kong) -
Định tuyến, cân bằng tải
│       ├── traefik.yml          # File cấu hình Traefik
│       └── Dockerfile           # Dockerfile cho API Gateway
├── devops/                      # Thư mục chứa cấu hình và script cho hoạt
động DevOps
│   ├── docker-compose.yml       # File Docker Compose cho môi trường phát
triển/kiểm thử cục bộ
│   ├── kubernetes/              # Thư mục chứa các Kubernetes manifests cho
môi trường production
│   │   ├── deployments/         # Định nghĩa các Deployment cho từng service
│   │   ├── services/            # Định nghĩa các Service để expose các Pod
│   │   ├── ingresses/           # Định nghĩa Ingress để quản lý truy cập
bên ngoài
│   │   ├── configmaps/          # Lưu trữ cấu hình không nhạy cảm
│   │   └── secrets/             # Lưu trữ thông tin nhạy cảm (mật khẩu, API
keys)
│   ├── monitoring/              # Cấu hình cho hệ thống giám sát và ghi log
│   │   ├── prometheus.yml       # Cấu hình Prometheus để thu thập metrics
│   │   ├── grafana-dashboards/  # Các dashboard Grafana để visualize metrics
│   │   └── elasticsearch/       # Cấu hình cho ELK Stack (Elasticsearch,
Logstash, Kibana)
│   └── scripts/                 # Các script tự động hóa triển khai, quản
lý, backup
├── .gitignore                   # File định nghĩa các file/thư mục bỏ qua
khi commit vào Git
├── README.md                    # File README tổng quan về dự án, hướng dẫn
cài đặt và chạy
└── LICENSE                      # File giấy phép phần mềm
```

Plain Text

```
![Sơ đồ khối tổng thể dự án](overall_system_block_diagram.png)

# Liên kết các sơ đồ để triển khai dự án
Để triển khai dự án Website Dịch vụ Toán học một cách hiệu quả và có hệ
thống, việc liên kết các sơ đồ (kiến trúc tổng thể, sơ đồ khối từng
microservice, cấu trúc thư mục) là rất quan trọng. Chúng cung cấp một cái
nhìn toàn diện từ cấp độ cao đến chi tiết, giúp đội ngũ phát triển và DevOps
hiểu rõ cách các thành phần hoạt động và tương tác với nhau.

## 1. Từ Sơ đồ Kiến trúc Tổng thể đến Sơ đồ Khối Microservice
```

*   **Sơ đồ Kiến trúc Tổng thể (`overall_system_block_diagram.png`):** Cung cấp cái nhìn tổng quan về toàn bộ hệ thống, các thành phần chính (Frontend, Backend Microservices, Databases, API Gateway, Message Broker) và mối quan hệ cấp cao giữa chúng. Nó giúp xác định ranh giới của từng Microservice và các hệ thống bên ngoài.

*   **Liên kết:** Mỗi

Microservice trong sơ đồ kiến trúc tổng thể (ví dụ: User Service, Payment Service) sẽ được chi tiết hóa bằng một **Sơ đồ khối Microservice riêng** (ví dụ: `user_service_block_diagram.png`, `payment_service_block_diagram.png`).

*   **Mục đích:** Sơ đồ khối của từng Microservice đi sâu vào cấu trúc bên trong của dịch vụ đó, chỉ ra các thành phần chính (API, logic nghiệp vụ, CRUD, kết nối DB, tích hợp bên ngoài) và luồng dữ liệu/tương tác giữa chúng. Điều này giúp các nhà phát triển hiểu rõ cách mỗi Microservice được xây dựng và hoạt động độc lập.

2. Từ Sơ đồ Khối Microservice đến Cấu trúc Thư mục

*   **Liên kết:** Mỗi thành phần trong sơ đồ khối của một Microservice (ví dụ: "User API", "User CRUD", "Authentication Logic" trong User Service) sẽ tương ứng với một hoặc nhiều thư mục/file cụ thể trong **Cấu trúc thư mục chi tiết của Microservice đó** (ví dụ: `user-service/app/api/`, `user-service/app/crud/`, `user-service/app/core/security.py`).

*   **Mục đích:** Cấu trúc thư mục cung cấp bản đồ vật lý của mã nguồn trên hệ thống file. Nó giúp các nhà phát triển dễ dàng tìm kiếm, tổ chức và quản lý code. Việc ánh xạ từ sơ đồ khối sang cấu trúc thư mục giúp đảm bảo rằng thiết kế được tuân thủ trong quá trình phát triển code.

3. Từ Cấu trúc Thư mục Microservice đến Cấu trúc Thư mục Tổng thể

*   **Liên kết:** Mỗi thư mục gốc của một Microservice (ví dụ: `user-service/`, `payment-service/`) sẽ là một thư mục con trong thư mục `backend/` của **Cấu trúc thư mục tổng thể toàn dự án** (`math-service-platform/backend/`). Tương tự, thư mục `frontend-web-app/` sẽ nằm trong `frontend/`.

*   **Mục đích:** Cấu trúc thư mục tổng thể cung cấp cái nhìn cấp cao về cách toàn bộ dự án được tổ chức trên hệ thống file. Nó rất quan trọng cho việc quản lý mã nguồn trong một kho lưu trữ (repository) lớn, cho các quy trình CI/CD (Continuous Integration/Continuous Deployment), và cho việc thiết lập môi trường phát triển cục bộ.

4. Vai trò của các sơ đồ trong triển khai

*   **Thiết kế và Phát triển:**
    *   **Sơ đồ Kiến trúc Tổng thể:** Giúp kiến trúc sư và đội ngũ quản lý có cái nhìn chiến lược, đảm bảo các Microservice được phân tách đúng đắn và giao tiếp hiệu quả.
    *   **Sơ đồ Khối Microservice:** Hướng dẫn các nhà phát triển xây dựng từng dịch vụ một cách độc lập, tập trung vào logic nghiệp vụ và các thành phần nội bộ.
    *   **Cấu trúc Thư mục:** Cung cấp khuôn mẫu cho việc viết code, đảm bảo tính nhất quán và dễ bảo trì trong toàn bộ dự án.

*   **Triển khai (Deployment):**
    *   **Sơ đồ Kiến trúc Tổng thể:** Hướng dẫn đội ngũ DevOps trong việc thiết lập hạ tầng (máy chủ, mạng, cân bằng tải, API Gateway, Message Broker).
    *   **Cấu trúc Thư mục:** Các `Dockerfile` trong mỗi Microservice và file `docker-compose.yml` (trong `devops/`) sử dụng cấu trúc thư mục này để xây dựng và chạy các container. Các Kubernetes manifests (trong `devops/kubernetes/`) cũng tham chiếu đến các thư mục này để triển khai các ứng dụng lên cluster.

*   **Vận hành và Giám sát (Operations & Monitoring):**
    *   **Sơ đồ Kiến trúc Tổng thể và Sơ đồ Khối Microservice:** Giúp đội ngũ vận hành hiểu luồng dữ liệu và các điểm tương tác quan trọng để thiết lập giám sát (Prometheus, Grafana) và ghi log tập trung (ELK Stack). Khi có sự cố, các sơ đồ này giúp nhanh chóng xác định Microservice hoặc thành phần bị ảnh hưởng.

*   **Mở rộng (Scaling):**
    *   **Sơ đồ Kiến trúc Tổng thể:** Cho phép xác định Microservice nào cần được mở rộng độc lập (ví dụ: Math Solver Service khi có nhiều yêu cầu giải toán).
    *   **Sơ đồ Khối Microservice:** Giúp hiểu rõ các thành phần bên trong có thể được tối ưu hóa hoặc mở rộng.

Bằng cách sử dụng các sơ đồ này một cách có hệ thống, dự án có thể được triển khai một cách rõ ràng, hiệu quả và dễ dàng quản lý trong suốt vòng đời phát triển.