ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN



KIỂM THỬ PHẦN MỀM

BÀI TẬP 3: THIẾT KẾ KIẾN TRÚC CHUẨN

Sinh viên : Mssv:

Trần Đức Toản 3122411219

Đỗ Minh Triết 3122411224

Nguyễn Việt Đức 3122411044

Phạm Văn Tính 3122411214

Giảng viên hướng dẫn: Đỗ Như Tài

Thành phố Hồ Chí Minh,Tháng 9 năm 2025

Mục Lục

[Phân công công việc](#_irl10f1ozjyc) 1

[Bài 1](#_n3n9tsk47jep) 2

[Bài 2](#_bt3ehxvb0691) 3

[Bài 3](#_5enzi8gasgpx) 4

[Bài 4](#_cw1giykltme3) 4

[Bài 5](#_rn05xvjlnnfe) 7

[Bài 6](#_tavwq9s32xve) 9

[Bài 7](#_1xe2y9eb4gtk) 11

[Bài tập ứng dụng:](#_m5d0l0b39013)**13**

[Sơ đồ C1](#_aaqe266s3fq6) 13

[Sơ đồ C2](#_lj04nlobbc11) 14

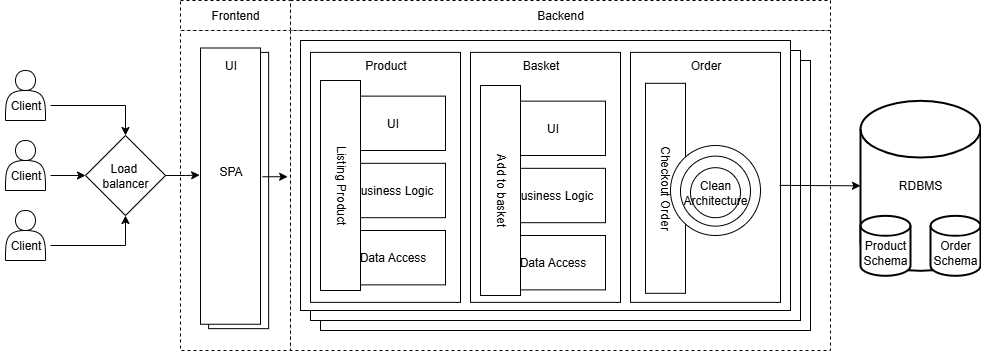
[Sơ đồ C3](#_wqxxc9k1k7wh) 16

[Sơ đồ C4](#_jhs0x51yf61t) 19

# Phân công công việc

| Họ tên | MSSV | Công việc | Hoàn thành |
| --- | --- | --- | --- |
| Trần Đức Toản | 3122411219 | Bài 3, Bài 4,sơ đồ c3 bài tập ứng dụng | 100% |
| Đỗ Minh Triết | 3122411224 | Bài 1, bài 2, sơ đồ c1 bài tập ứng dụng | 100% |
| Nguyễn Việt Đức | 3122411044 | Bài 5, bài 6, sơ đồ c2 bài tập ứng dụng | 100% |
| Phạm Văn Tính | 3122411214 | Bài 7, bài 8, sơ đồ c4 bài tập ứng dụng | 100% |

# Bài 1



Giải thích :

Kiến trúc của một hệ thống web sử dụng SPA ( Single Page Application ) và các Microservices

Client : Các người dùng hệ thống

Load Balancer : Phân tán các yêu cầu từ người dùng từ frontend

Frontend : Hiện UI mà người dùng thấy, sử dụng SPA - Trang web sẽ load một lần và cập nhật khi có thay đổi mà không cần load lại từ đầu

Backend : Gồm nhiều các services khác nhau

* Product :
  + UI : Hiện sản phẩm
  + BL : Xử lý các yêu cầu nghiệp vụ cho sản phẩm
  + DA : Giao tiếp với DB

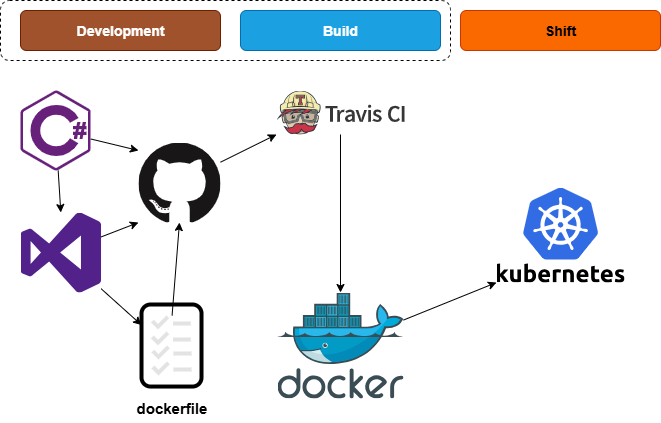
Liệt kê các sản phẩm

* Basket :
  + UI : Hiện thông tin giỏ hàng
  + BL : Xử lý các yêu cầu nghiệp vụ cho giỏ hàng
  + DA : Giao tiếp với DB

Thêm sản phẩm vào giỏ hàng

* Order : Xử lý các yêu cầu nghiệp vụ cho sản phẩm
  + Sử dụng clean architecture để đảm bảo tính linh hoạt, toàn vẹn, dễ bảo trì
* RDBMS : CSDL lưu trữ các lược đồ cho các services

# Bài 2



Giải thích :

Các bước xây dựng từ lúc phát triển tới khi deploy :

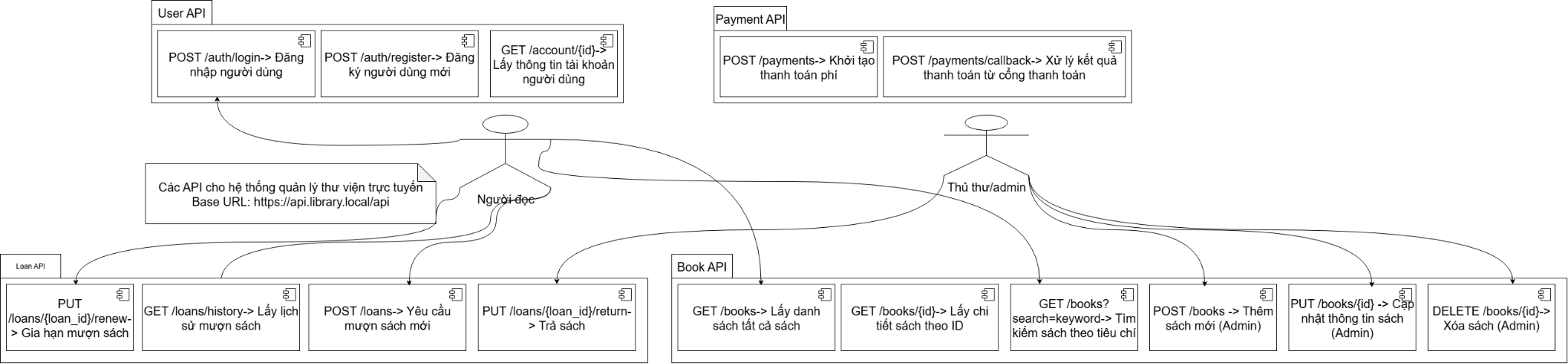
Ứng dụng được phát triển sử dụng ngôn ngữ C# qua các phần mềm như Visual Studio , Github. Github được dùng dể lưu các bản thay đổi còn Visual Studio là phần mềm để lập trình.

Dockerfile được tạo để lưu lại cách đóng gói ứng dụng.

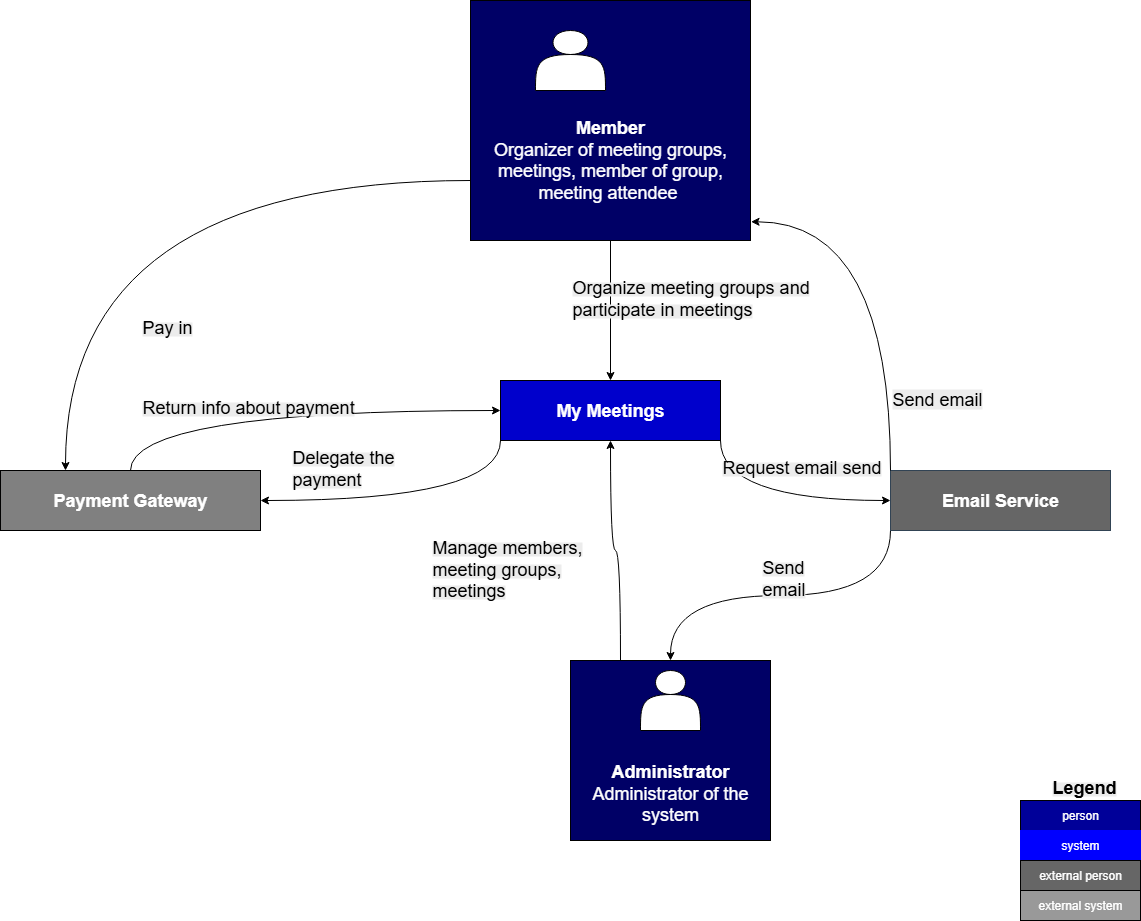
Tiếp theo Travis CI lấy code từ Github qua các lần push và build ứng dụng và chạy test và tạo Docker Image

Kubernetes lấy Docker Image để schedule nhiều container cho ứng dụng

# Bài 3



# Bài 4



Giải thích

1, Hệ Thống Chính (System under consideration)

* Tên: My Meetings
* Mô tả: Đây là hệ thống phần mềm mà chúng ta đang xây dựng. Dựa vào các tương tác, có thể thấy "My Meetings" là một nền tảng cho phép người dùng tổ chức và tham gia vào các cuộc họp hoặc các nhóm họp. Hệ thống này quản lý thông tin về thành viên, nhóm, cuộc họp và xử lý các nghiệp vụ liên quan.

Các Tác Nhân (Persons / Actors)

Đây là những người sẽ trực tiếp sử dụng hệ thống.

Member (Thành viên):

* Vai trò: Là người dùng cuối của hệ thống. Họ có thể là người tổ chức cuộc họp, người tạo nhóm, hoặc chỉ đơn giản là thành viên của một nhóm và người tham dự cuộc họp.
* Tương tác chính: Họ sử dụng hệ thống "My Meetings" để tổ chức các nhóm họp và tham gia các cuộc họp. Họ cũng thực hiện thanh toán qua Cổng thanh toán và nhận email từ Dịch vụ Email.

Administrator (Quản trị viên):

* Vai trò: Là người quản lý, vận hành toàn bộ hệ thống "My Meetings".
* Tương tác chính: Họ sử dụng hệ thống để quản lý thành viên, các nhóm họp, và các cuộc họp. Họ có quyền hạn cao hơn "Member" và chịu trách nhiệm giữ cho hệ thống hoạt động trơn tru.

2,Các Hệ Thống Bên Ngoài (External Systems)

Đây là các hệ thống phần mềm khác mà "My Meetings" phụ thuộc vào để hoàn thành chức năng của mình.

Payment Gateway (Cổng Thanh Toán):

* Vai trò: Là một hệ thống của bên thứ ba chuyên xử lý các giao dịch tài chính (ví dụ: thanh toán bằng thẻ tín dụng, ví điện tử).
* Tương tác: "My Meetings" không tự xử lý thanh toán (đây là một thực hành tốt về bảo mật), thay vào đó, nó ủy thác việc thanh toán cho Payment Gateway. Sau khi người dùng thanh toán, Payment Gateway sẽ trả lại thông tin về giao dịch (thành công hay thất bại) cho "My Meetings".

Email Service (Dịch vụ Email):

* Vai trò: Là một hệ thống chuyên dụng cho việc gửi email với số lượng lớn, đảm bảo email đến được hộp thư của người nhận (ví dụ: Amazon SES, SendGrid).
* Tương tác: Khi cần gửi thông báo, lời mời họp, hoặc các email khác, "My Meetings" sẽ yêu cầu gửi email tới dịch vụ này. Email Service sẽ thay mặt "My Meetings" để gửi email đến cho "Member" hoặc các người dùng khác.

3. Luồng Tương Tác Chính

Dựa trên các mũi tên, chúng ta có thể hình dung ra các luồng hoạt động chính:

Luồng Tổ Chức Một Cuộc Họp Có Phí:

* Member đăng nhập vào My Meetings và tạo một cuộc họp.
* Khi đến bước thanh toán, My Meetings chuyển hướng người dùng sang Payment Gateway.
* Member thực hiện việc thanh toán (Pay in) trực tiếp trên giao diện của Payment Gateway.
* Payment Gateway xử lý và trả về thông tin thanh toán cho My Meetings.
* My Meetings dựa vào kết quả để cập nhật trạng thái cuộc họp.

Luồng Gửi Lời Mời Họp:

* Một Member (người tổ chức) sử dụng My Meetings để mời các thành viên khác.
* My Meetings tạo một yêu cầu gửi email đến Email Service.
* Email Service nhận yêu cầu và gửi email lời mời đến hộp thư của những người tham dự.

Luồng Quản Trị Hệ Thống:

* Administrator đăng nhập vào My Meetings.
* Họ thực hiện các hành động như xem danh sách thành viên, xóa một cuộc họp không hợp lệ, hoặc cấu hình hệ thống. Tất cả các hành động này được mô tả chung là "Manage members, meeting groups, meetings".

# Bài 5

Giải thích:Đây là một Sơ đồ Kiến trúc C4 cấp độ 2 (Container Diagram), cung cấp cái nhìn tổng quan về cấu trúc cao cấp của “My Meetings System”, làm rõ các ứng dụng chính (containers) và cách chúng tương tác với người dùng và các hệ thống bên ngoài.

Hệ thống hoạt động theo một chuỗi tương tác rõ ràng:

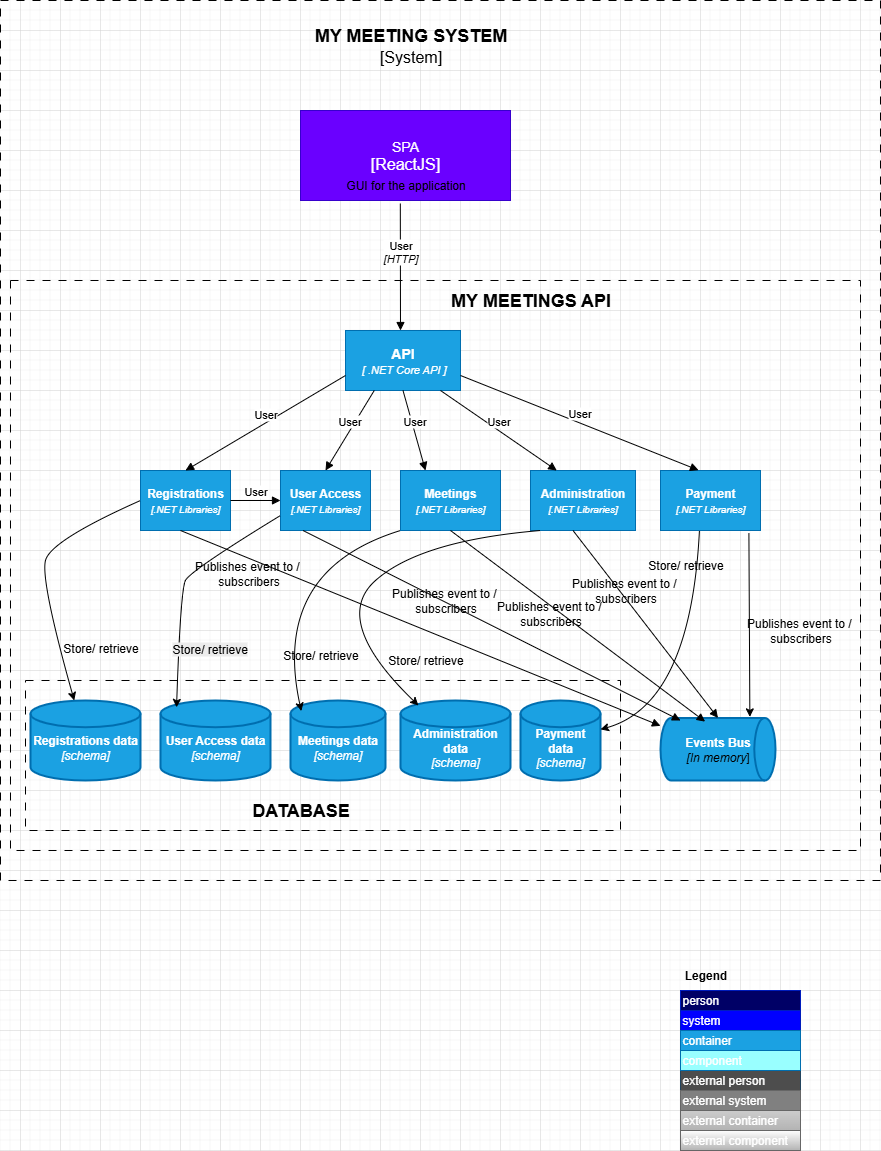
* Người dùng truy cập giao diện:
  + Có hai loại người dùng chính: Member (Thành viên), người tạo hoặc tham gia các cuộc họp, và Administrator (Quản trị viên), người quản lý hệ thống.
  + Cả hai đều truy cập vào hệ thống thông qua SPA [ReactJS] (Single Page Application - Ứng dụng một trang được xây dựng bằng ReactJS). Đây là giao diện người dùng (GUI) chạy trên trình duyệt web của họ.
* SPA giao tiếp với Backend:
  + Khi người dùng thực hiện một hành động trên SPA (ví dụ: tạo cuộc họp, xem danh sách thành viên), SPA sẽ gửi các yêu cầu thông qua giao thức HTTP đến My Meetings API [.NET Core] Backend.
  + Phần Backend này được xây dựng bằng .NET Core và chứa toàn bộ logic nghiệp vụ, xử lý các yêu cầu từ SPA.
* Backend xử lý dữ liệu và tích hợp dịch vụ:
  + Lưu trữ dữ liệu: Backend giao tiếp với Database [Microsoft SQL] (cơ sở dữ liệu Microsoft SQL) để đọc và ghi dữ liệu liên quan đến các cuộc họp, nhóm, thành viên, v.v., sử dụng các câu lệnh SQL.
  + Thanh toán: Khi người dùng (thông qua SPA và Backend) cần thực hiện thanh toán, Backend sẽ "ủy quyền thanh toán" thông qua giao thức HTTP cho một Payment Gateway (Cổng thanh toán) bên thứ ba. Backend không tự xử lý logic thanh toán mà dựa vào dịch vụ chuyên biệt này.
  + Email: Backend cũng sử dụng một Email System (Hệ thống Email) bên thứ ba (ví dụ: một máy chủ SMTP) để gửi email (ví dụ: xác nhận đăng ký, thông báo, nhắc nhở cuộc họp) thông qua giao thức SMTP.

Tóm tắt luồng điển hình:

* Người dùng (Member/Admin) truy cập ứng dụng web (SPA).
* SPA gọi API backend để lấy/gửi dữ liệu.
* API backend tương tác với cơ sở dữ liệu để lưu/truy xuất thông tin.
* Nếu có thanh toán, API backend gọi cổng thanh toán bên thứ ba.
* Nếu cần gửi email, API backend gọi hệ thống email bên thứ ba.

Sơ đồ Container này có ý nghĩa quan trọng trong việc cung cấp một cái nhìn tổng thể về kiến trúc "My Meetings System". Nó giúp làm rõ cấu trúc hệ thống và phân định trách nhiệm của từng ứng dụng/dịch vụ chính (container), đồng thời minh bạch về công nghệ được sử dụng (ReactJS, .NET Core, SQL). Đặc biệt, nó nhấn mạnh các điểm tích hợp quan trọng với các dịch vụ bên ngoài (thanh toán, email), từ đó tạo cơ sở vững chắc cho việc truyền đạt, thảo luận và lập kế hoạch phát triển, triển khai cũng như bảo trì hệ thống cho tất cả các bên liên quan.

# Bài 6



Giải thích:

Sơ đồ này là một Component Diagram (C4 cấp độ 3), trình bày cấu trúc chi tiết bên trong của My Meetings API. Nó "phóng to" phần API để hiển thị các Thành phần (.NET Libraries) và cách chúng hoạt động.

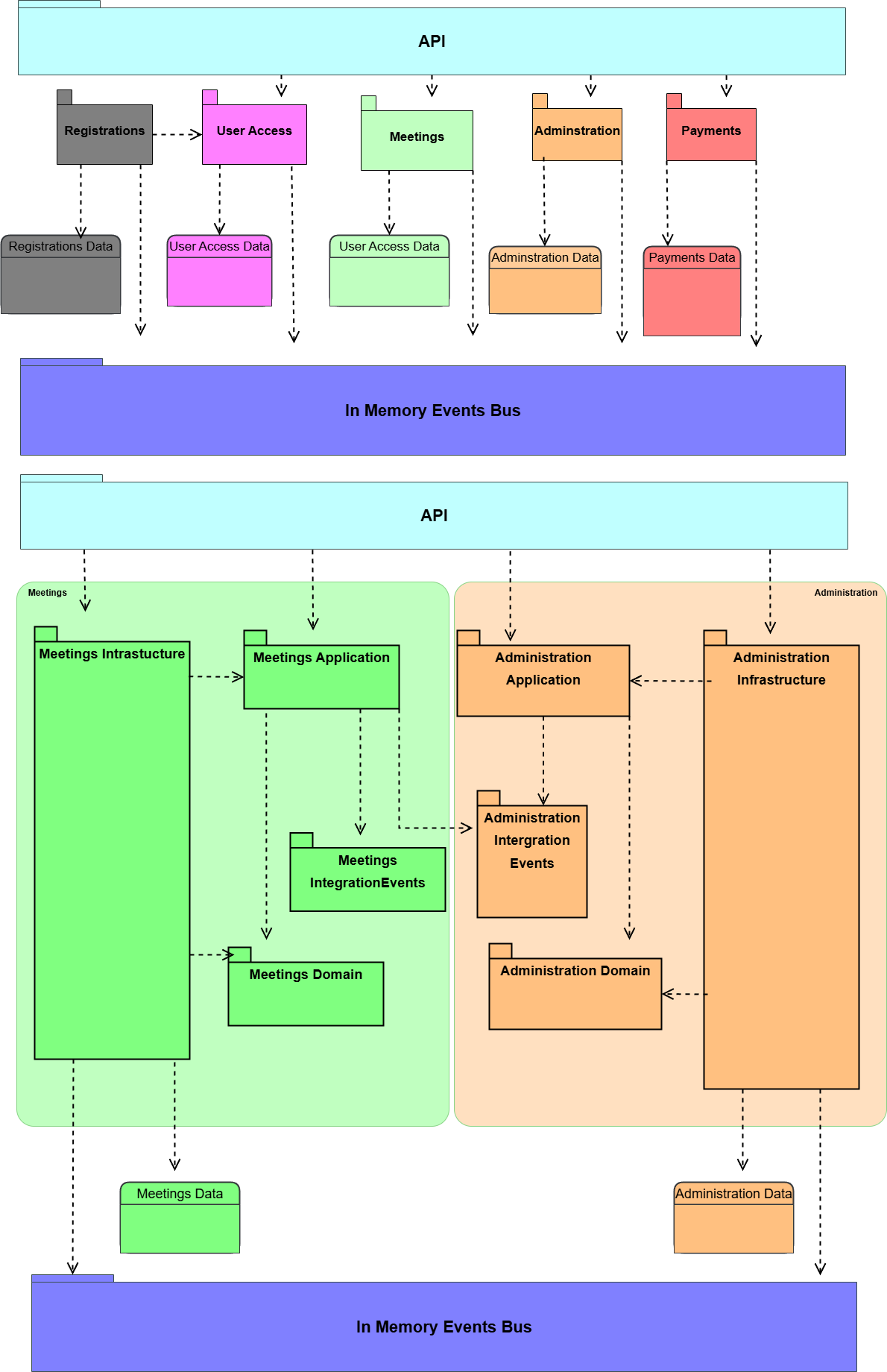
Cơ chế hoạt động:

1. SPA (ReactJS) gửi yêu cầu HTTP đến API Gateway (.NET Core API).
2. API Gateway điều hướng yêu cầu đến các Thành phần chức năng chuyên biệt (Registrations, User Access, Meetings, Administration, Payment).
3. Mỗi Thành phần lưu/truy xuất dữ liệu từ các schema chuyên biệt trong Database.
4. Các Thành phần cũng xuất bản hoặc đăng ký nhận sự kiện thông qua Events Bus (trong bộ nhớ) để giao tiếp nội bộ một cách linh hoạt.

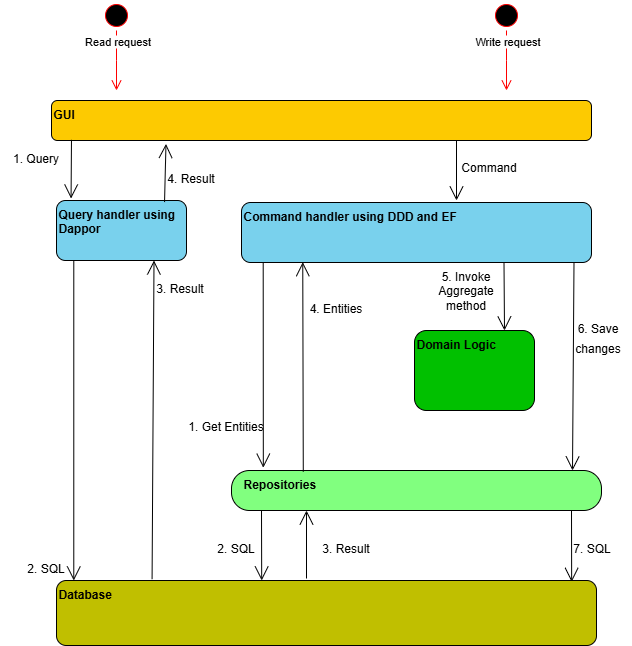
Ý nghĩa:

Sơ đồ này cho thấy cách backend được tổ chức thành các module chức năng rõ ràng, sử dụng kiến trúc hướng sự kiện nội bộ để giao tiếp và phân vùng dữ liệu tương ứng. Nó giúp hiểu sâu hơn về thiết kế backend, hỗ trợ phát triển, bảo trì và mở rộng hệ thống.

# Bài 7



Bài 8



# Bài tập ứng dụng:

## Sơ đồ C1

## 

Sơ đồ Context Diagram trên thể hiện dòng chảy dữ liệu cơ bản của một thư viện online với 3 tác nhân chính giao tiếp với hệ thống như sau :

Người dùng :

* Người mượn sách thực hiện mượn , trả hay gia hạn việc mượn sách rồi nhận các phiếu mượn , trả , gia hạn.
* Tìm kiếm , xem lịch sử mượn và xem từ hệ thống.
* Thanh toán chi phí thành viên, phí trễ hạn thông qua thông tin nhập vào hệ thống

Thủ thư :

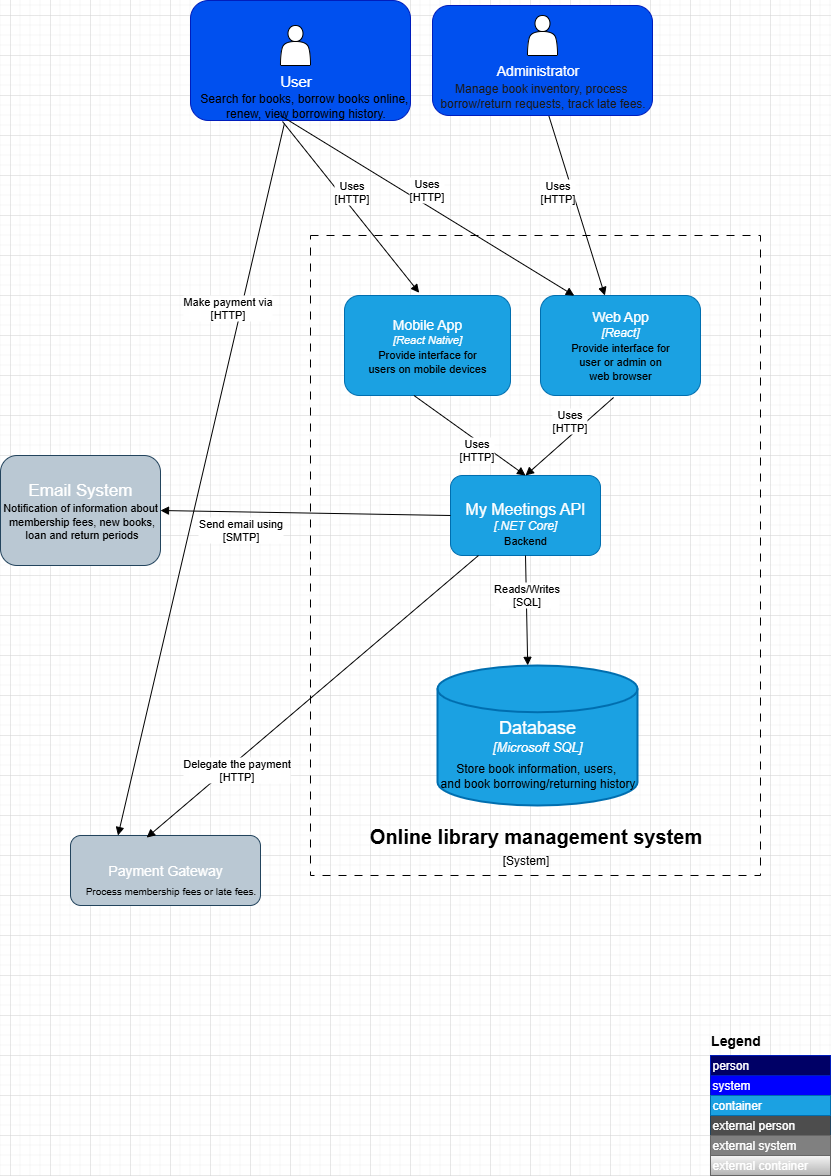
* Kiểm tra và xử lý các phiếu mượn trả , gia hạn được cung cấp từ hệ thống
* Quản lý kho sách bằng hệ thống thư viện online
* Theo dõi thông tin trễ hạn của người dùng qua hệ thống

Hệ thống thanh toán : Các cổng thanh toán ngoài hệ thống

* Nhận thông tin thanh toán từ hệ thống thư viện và kiểm tra và thanh toán nếu được

# 

## Sơ đồ C2



Câu hỏi 1: Tại sao bạn lại xây dựng cả Web App và Mobile App riêng biệt?

Trả lời: Việc này nhằm phục vụ tốt nhất cho hai nhóm đối tượng sử dụng khác nhau:

* Web App: Chủ yếu dành cho Quản trị viên (Administrator) với các tác vụ phức tạp như quản lý kho sách, xem báo cáo, xử lý yêu cầu... cần một giao diện đầy đủ và thao tác bằng chuột, bàn phím trên màn hình lớn.
* Mobile App: Hướng đến Người dùng (User), mang lại sự tiện lợi tối đa. Họ có thể dễ dàng tìm kiếm sách, gia hạn, xem lịch sử mượn ở bất cứ đâu. Ngoài ra, ứng dụng di động còn có thể tận dụng các tính năng của điện thoại như thông báo đẩy (push notification) để nhắc nhở hạn trả sách.

Câu hỏi 2: Tại sao bạn chọn React cho Web App và React Native cho Mobile App?

Trả lời: Lựa chọn này dựa trên hiệu quả và sự đồng bộ:

* Trải nghiệm người dùng: React giúp xây dựng ứng dụng trang đơn (Single Page Application - SPA), cho phép tải nội dung động mà không cần tải lại cả trang. Điều này tạo ra một trải nghiệm web rất nhanh và mượt mà.
* Hiệu quả phát triển: Vì React và React Native cùng chung một hệ sinh thái, đội ngũ phát triển có thể chia sẻ một phần đáng kể mã nguồn, đặc biệt là các logic quản lý trạng thái và xử lý dữ liệu, giúp giảm thời gian và công sức phát triển cho cả hai nền tảng.

Câu hỏi 3 : Tại sao bạn chọn .NET Core để xây dựng API?

Trả lời: .NET Core là một lựa chọn mạnh mẽ cho backend vì:

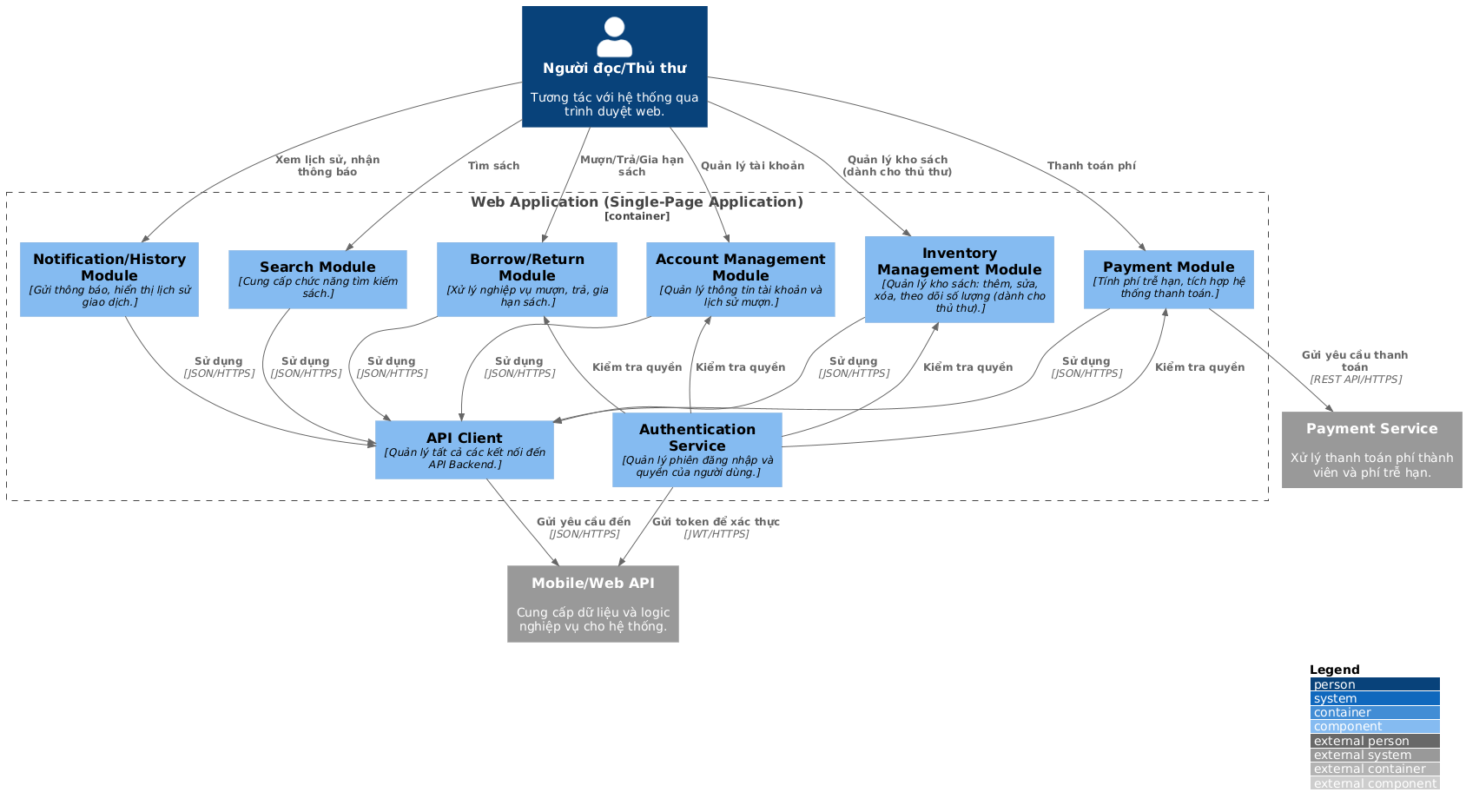
* Hiệu năng vượt trội: Nó được thiết kế để xử lý số lượng lớn yêu cầu đồng thời một cách hiệu quả, đảm bảo hệ thống luôn phản hồi nhanh ngay cả khi có nhiều người dùng truy cập.
* Hệ sinh thái và công cụ hỗ trợ tốt: .NET Core được hỗ trợ bởi Microsoft, đi kèm với các công cụ mạnh mẽ như Entity Framework Core giúp việc tương tác với cơ sở dữ liệu trở nên đơn giản và an toàn hơn rất nhiều.
* Linh hoạt trong triển khai: API xây dựng bằng .NET Core có thể chạy trên nhiều hệ điều hành (Windows, Linux), mang lại sự tự do trong việc lựa chọn môi trường máy chủ để tối ưu chi phí.

Câu hỏi 4 : Tại sao bạn quyết định tích hợp với Payment Gateway (Cổng thanh toán) và Email System bên ngoài thay vì tự xây dựng?

Trả lời: Quyết định này dựa trên nguyên tắc "tập trung vào giá trị cốt lõi":

* An toàn và chuyên môn hóa: Xử lý thanh toán và gửi email là những lĩnh vực rất phức tạp. Các nhà cung cấp dịch vụ chuyên dụng đã giải quyết các vấn đề khó khăn như bảo mật theo tiêu chuẩn quốc tế (PCI DSS cho thanh toán) và đảm bảo email không bị vào mục spam. Tự xây dựng sẽ rất tốn kém và rủi ro.
* Tập trung nguồn lực: Bằng cách giao phó các tác vụ này cho bên thứ ba, đội ngũ phát triển có thể dành toàn bộ thời gian và tâm huyết để xây dựng và cải tiến các tính năng chính của hệ thống quản lý thư viện, vốn là giá trị thực sự mà sản phẩm mang lại cho người dùng.

## Sơ đồ C3



Giải thích các Quyết định Thiết kế chính

Tại sao lại là Single-Page Application (SPA)?

Lý do lựa chọn :

* Trải nghiệm người dùng liền mạch: Đây là lợi ích lớn nhất. Toàn bộ trang web chỉ tải một lần. Sau đó, khi người dùng điều hướng giữa các chức năng (ví dụ: từ tìm kiếm sang xem lịch sử mượn), chỉ có phần nội dung cần thiết được tải về từ API và cập nhật động. Điều này tạo ra cảm giác sử dụng mượt mà, nhanh chóng, gần giống như một ứng dụng trên máy tính (desktop application).
* Tách biệt hoàn toàn Frontend và Backend: Kiến trúc này επιβάλλει một sự tách biệt rõ ràng. Đội Frontend chỉ cần tập trung vào giao diện và trải nghiệm người dùng, giao tiếp với Backend thông qua một bộ API đã được định nghĩa trước. Đội Backend cũng có thể phát triển độc lập mà không cần quan tâm giao diện được xây dựng bằng công nghệ gì.
* Tối ưu hóa cho giao tiếp API: SPA được thiết kế để "nói chuyện" hiệu quả với các API, gửi và nhận dữ liệu (thường là định dạng JSON) một cách gọn nhẹ, giúp giảm tải cho máy chủ và băng thông.

Tại sao cần có thành phần API Client riêng biệt?

Lý do lựa chọn :

* Tuân thủ nguyên tắc DRY (Don't Repeat Yourself): Toàn bộ logic lặp đi lặp lại như cấu hình URL cơ sở của API, thêm Access Token vào header để xác thực, xử lý lỗi chung (401, 403, 500)... được tập trung tại một nơi duy nhất. Các module khác không cần phải viết lại logic này.
* Dễ dàng bảo trì và thay đổi: Nếu sau này hệ thống thay đổi cơ chế xác thực hoặc chuyển sang một phiên bản API mới (ví dụ từ /v1 lên /v2), chúng ta chỉ cần cập nhật logic ở duy nhất file API Client là xong.
* Tăng khả năng kiểm thử (Testability): Khi viết Unit Test cho một module nghiệp vụ, chúng ta có thể dễ dàng "giả lập" (mock) API Client để cung cấp dữ liệu giả, giúp việc kiểm thử logic của module trở nên độc lập và đáng tin cậy mà không cần phụ thuộc vào API thật.

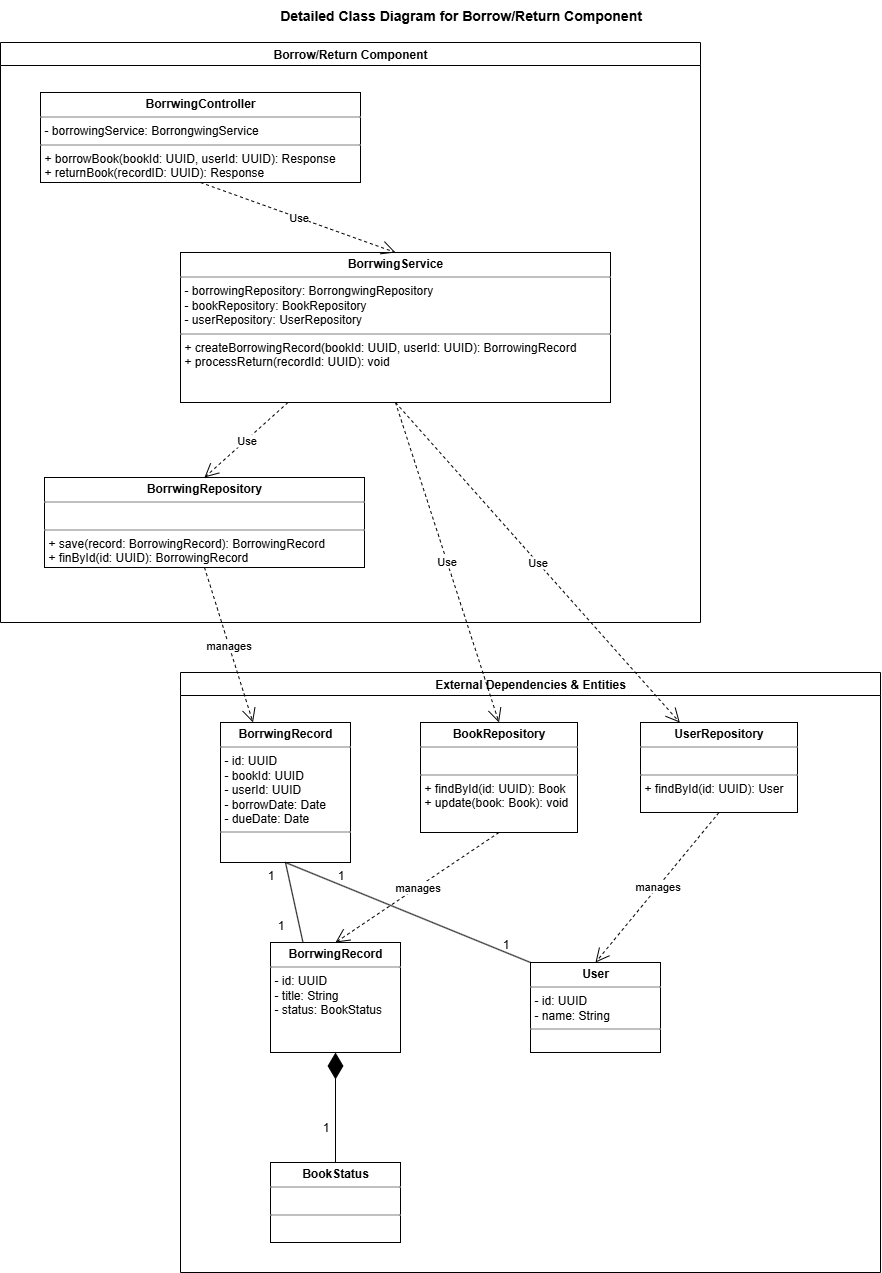
Tại sao chia thành các Module nhỏ (Search, Borrow, Account)?

Lý do lựa chọn :

* Dễ quản lý và dễ hiểu: Mỗi module có một trách nhiệm rõ ràng. Một lập trình viên mới có thể nhanh chóng hiểu được cấu trúc dự án và tìm thấy đoạn code họ cần sửa. Điều này giúp giảm "chi phí nhận thức" (cognitive overhead).
* Thúc đẩy phát triển song song: Các nhóm khác nhau có thể làm việc trên các module khác nhau cùng một lúc mà ít gây ra xung đột code. Team A có thể tập trung phát triển tính năng mượn/trả trong khi Team B cải thiện chức năng tìm kiếm.
* Tăng khả năng mở rộng (Scalability): Khi hệ thống cần thêm một chức năng lớn mới (ví dụ: quản lý sự kiện trong thư viện), chúng ta có thể tạo một module mới mà không gây ảnh hưởng lớn đến các module hiện có.

.

## Sơ đồ C4



**Tổng quan về Sơ đồ**

Sơ đồ mô tả kiến trúc chi tiết ở mức code cho thành phần "Mượn/Trả sách" (Borrow/Return Component). Nó được thiết kế theo mô hình 3 lớp rất phổ biến và hiệu quả, bao gồm: Controller, Service, và Repository.

**Diễn giải chi tiết từng lớp**

**1. Lớp Controller (Tầng giao tiếp)**

* Lớp: BorrowingController
* Nhiệm vụ: Đây là lớp cổng vào, chịu trách nhiệm tiếp nhận các yêu cầu trực tiếp từ người dùng (ví dụ: thông qua một yêu cầu HTTP khi người dùng bấm nút "Mượn sách").
* Cách hoạt động: Nó không tự xử lý logic nghiệp vụ. Thay vào đó, nó gọi đến lớp BorrowingService để thực hiện công việc chính.
* Ví dụ: Khi người dùng yêu cầu mượn sách, borrowBook() sẽ được gọi. Lớp này chỉ đơn giản là chuyển tiếp bookId và userId cho BorrowingService.

**2. Lớp Service (Tầng nghiệp vụ)**

* Lớp: BorrowingService
* Nhiệm vụ: Đây là "bộ não" của thành phần, chứa toàn bộ logic nghiệp vụ phức tạp.
* Cách hoạt động: Nó điều phối hoạt động giữa các Repository để thực hiện một quy trình hoàn chỉnh.
* Ví dụ: Khi được yêu cầu tạo một lượt mượn sách (createBorrowingRecord), BorrowingService sẽ:
  1. Dùng BookRepository để tìm và kiểm tra trạng thái của sách.
  2. Dùng UserRepository để xác thực người mượn.
  3. Nếu mọi thứ hợp lệ, nó sẽ tạo một đối tượng BorrowingRecord mới.
  4. Cuối cùng, nó dùng BorrowingRepository để lưu lượt mượn này vào cơ sở dữ liệu và dùng BookRepository để cập nhật lại trạng thái sách (ví dụ: từ AVAILABLE sang BORROWED).

**3. Lớp Repository (Tầng truy cập dữ liệu)**

* Các lớp: BorrowingRepository, BookRepository, UserRepository
* Nhiệm vụ: Đây là lớp duy nhất được phép giao tiếp trực tiếp với cơ sở dữ liệu. Mỗi Repository quản lý một loại dữ liệu cụ thể.
* Cách hoạt động: Chúng cung cấp các phương thức đơn giản để các lớp Service có thể lấy hoặc lưu dữ liệu mà không cần biết về chi tiết cách dữ liệu được lưu trữ (ví dụ: không cần biết về câu lệnh SQL).
* Ví dụ: BookRepository có phương thức findById() để lấy thông tin một cuốn sách và update() để cập nhật nó.

**4. Các Entities (Đối tượng dữ liệu)**

* Các lớp: BorrowingRecord, Book, User, BookStatus
* Nhiệm vụ: Đây là các lớp đơn giản, chỉ dùng để biểu diễn và lưu trữ dữ liệu. Chúng tương ứng với các bảng trong cơ sở dữ liệu.
* Ví dụ: Book chứa các thông tin như id, title, và status. BookStatus là một enum định nghĩa các trạng thái cố định mà một cuốn sách có thể có.

**Các quyết định thiết kế chính ở Mức C4 (Sơ đồ Code)**

**1. Tại sao áp dụng kiến trúc 3 lớp: Controller-Service-Repository?**

* Quyết định: Cấu trúc code được chia thành ba lớp trách nhiệm rõ ràng: BorrowingController, BorrowingService, và các lớp Repository.
* Lý do lựa chọn: Đây là một quyết định nền tảng dựa trên nguyên tắc Phân tách trách nhiệm (Separation of Concerns). Mỗi lớp chỉ làm một việc duy nhất và làm tốt việc đó:
  + Controller chỉ giao tiếp với bên ngoài.
  + Service chỉ chứa logic nghiệp vụ.
  + Repository chỉ truy cập dữ liệu.
* Lợi ích:
  + Dễ kiểm thử (Testability): Có thể kiểm thử logic nghiệp vụ trong BorrowingService một cách độc lập mà không cần đến cơ sở dữ liệu hay giao diện người dùng.
  + Dễ bảo trì: Khi cần thay đổi logic nghiệp vụ, bạn chỉ cần sửa ở lớp Service. Khi cần thay đổi cách truy cập cơ sở dữ liệu, bạn chỉ cần sửa ở lớp Repository. Các thay đổi ít ảnh hưởng lẫn nhau.

**2. Tại sao sử dụng các lớp Entity đơn giản (POCO - Plain Old CLR Object)?**

* Quyết định: Các lớp như Book, User, và BorrowingRecord được thiết kế chỉ để chứa dữ liệu, không chứa các logic phức tạp.
* Lý do lựa chọn: Quyết định này giúp tách biệt hoàn toàn mô hình dữ liệu khỏi hạ tầng truy cập dữ liệu. Các lớp Entity này không cần "biết" chúng được lưu vào cơ sở dữ liệu như thế nào.
* Lợi ích: Giúp cho lớp Service có thể hoạt động với các đối tượng dữ liệu thuần túy, làm cho logic nghiệp vụ trở nên trong sáng và không bị phụ thuộc vào một công nghệ cơ sở dữ liệu cụ thể.

**3. Tại sao phân tách Repository cho từng Entity?**

* Quyết định: Thay vì có một lớp truy cập dữ liệu khổng lồ, kiến trúc này sử dụng một Repository riêng cho mỗi Entity chính: BookRepository, UserRepository, BorrowingRepository.
* Lý do lựa chọn: Tuân thủ Nguyên tắc trách nhiệm đơn nhất (Single Responsibility Principle). Mỗi lớp Repository chỉ có một lý do duy nhất để thay đổi: khi cấu trúc của Entity tương ứng thay đổi.
* Lợi ích: Giúp cho mã nguồn truy cập dữ liệu được tổ chức gọn gàng, tránh việc một lớp trở nên quá lớn và khó quản lý.