**BÁO CÁO BÀI TẬP LAB 3**

**KIỂM THỬ PHẦN MỀM**

**GV hướng dẫn: TS. Đỗ Như Tài**

**Lớp: DCT122C3 – Mã học phần: 841408**

**Sinh viên: Trịnh Long Phát – 3122411150**

**Lê Hồng Phát – 3122411145**

**Trương Phú Kiệt – 3122411109**

**Trà Đức Toàn – 3122411217**

**Thành phố Hồ Chí Minh, tháng 9 năm 2025**

**BẢNG PHÂN CÔNG CỦA NHÓM**

|  |  |  |
| --- | --- | --- |
| **Tên các thành viên thực hiện** | **Nội dung công việc** | **Tiến độ**  **công việc** |
| **Trịnh Long Phát** | **Bài tập minh họa: bài 1, 7** | **100%** |
| **Lê Hồng Phát** | **Tổng hợp + thiết kế word** | **100%** |
| **Bài tập minh họa: bài 2, 8** | **100%** |
| **Trương Phú Kiệt** | **Bài tập minh họa: 5, 6** | **100%** |
| **Bài tập ứng dụng** | **100%** |
| **Trà Đức Toàn** | **Bài tập minh họa: bài 3, 4** | **100%** |

**MỤC LỤC**

[BÀI TẬP MINH HỌA 4](#_Toc209544481)

[Bài 1. Vẽ lại sơ đồ kiến trúc SPA (Single Page Application): 4](#_Toc209544482)

[Bài 2. Vẽ sơ đồ triển khai CI/CD: 4](#_Toc209544483)

[Bài 3. Vẽ sơ đồ API của hệ thống: 5](#_Toc209544484)

[Bài 4. Vẽ sơ đồ C1 - System Context và giải thích: 6](#_Toc209544485)

[Bài 5. Vẽ sơ đồ C2 – Container và giải thích: 9](#_Toc209544486)

[Bài 6. Vẽ sơ đồ C3 – Component (High-Level) và giải thích: 11](#_Toc209544487)

[Bài 7. Vẽ sơ đồ C3 – Component (Module-Level) sau: 13](#_Toc209544488)

[Bài 8. Vẽ sơ đồ xử lý 1 request: 15](#_Toc209544489)

[BÀI TẬP ỨNG DỤNG 19](#_Toc209544490)

[Tổng quan kiến trúc 20](#_Toc209544491)

[Level 1 (Context Diagram): Hệ thống và các tác nhân bên ngoài. 21](#_Toc209544492)

[Level 2 (Container Diagram): Các container (ứng dụng web, mobile app, database, service...). 22](#_Toc209544493)

[Level 3 (Component Diagram): Các thành phần trong một container quan trọng. 23](#_Toc209544494)

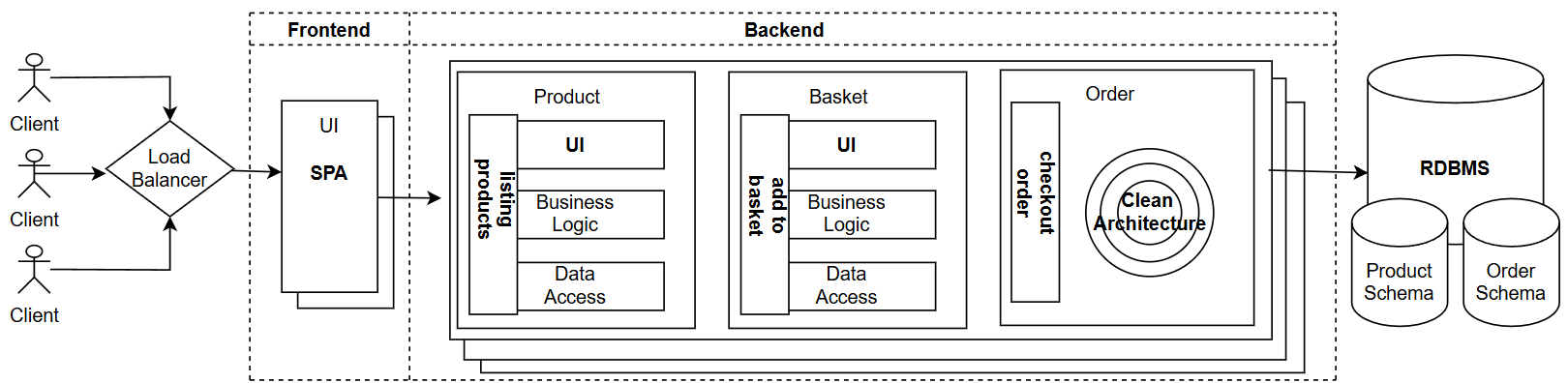
[Level 4 (Code/Implementation Diagram): Chi tiết lớp hoặc cấu trúc code. 24](#_Toc209544495)

THIẾT KẾ KIẾN TRÚC CHUẨN C4 MODEL

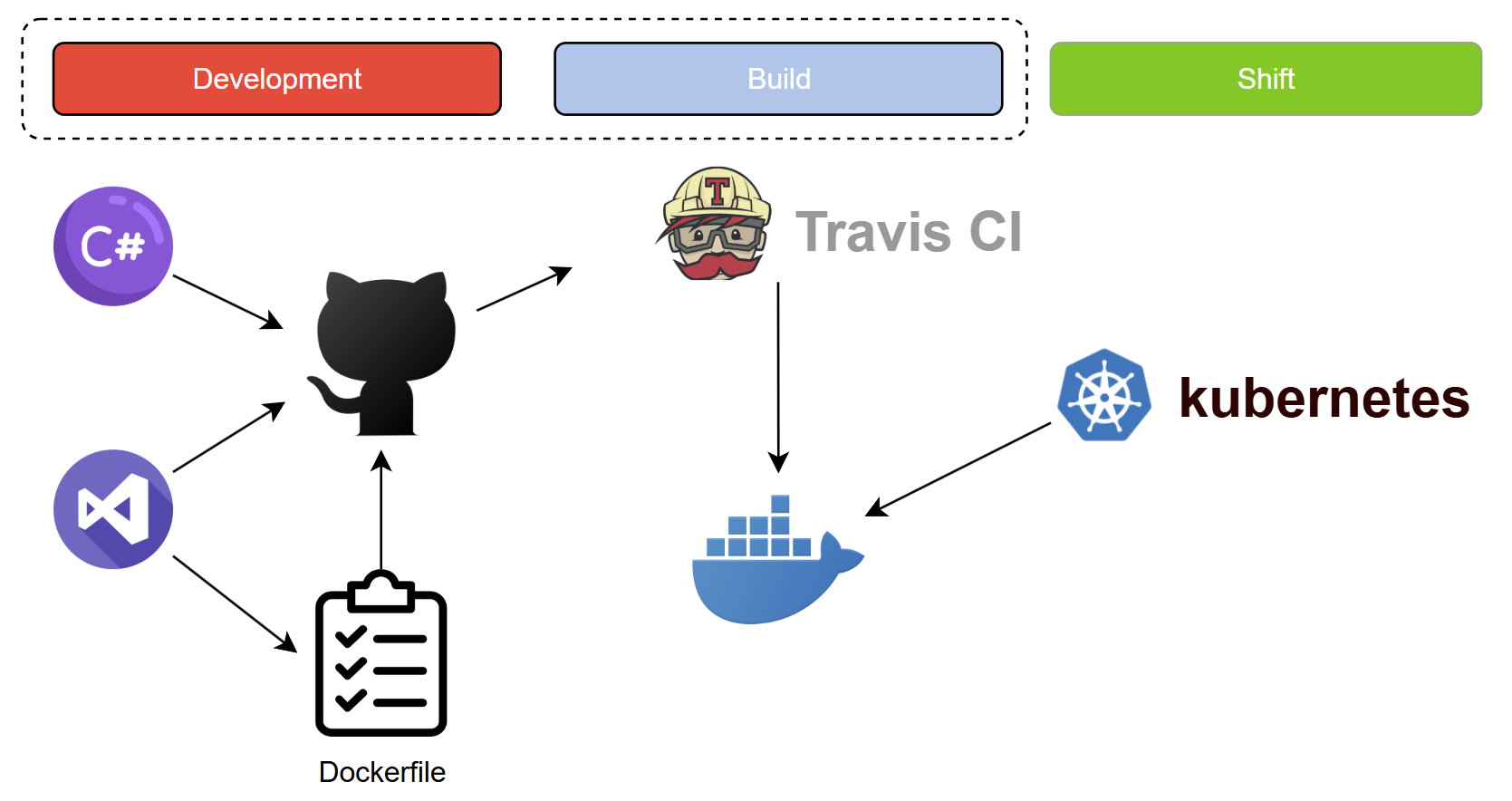
---o0o---

# BÀI TẬP MINH HỌA

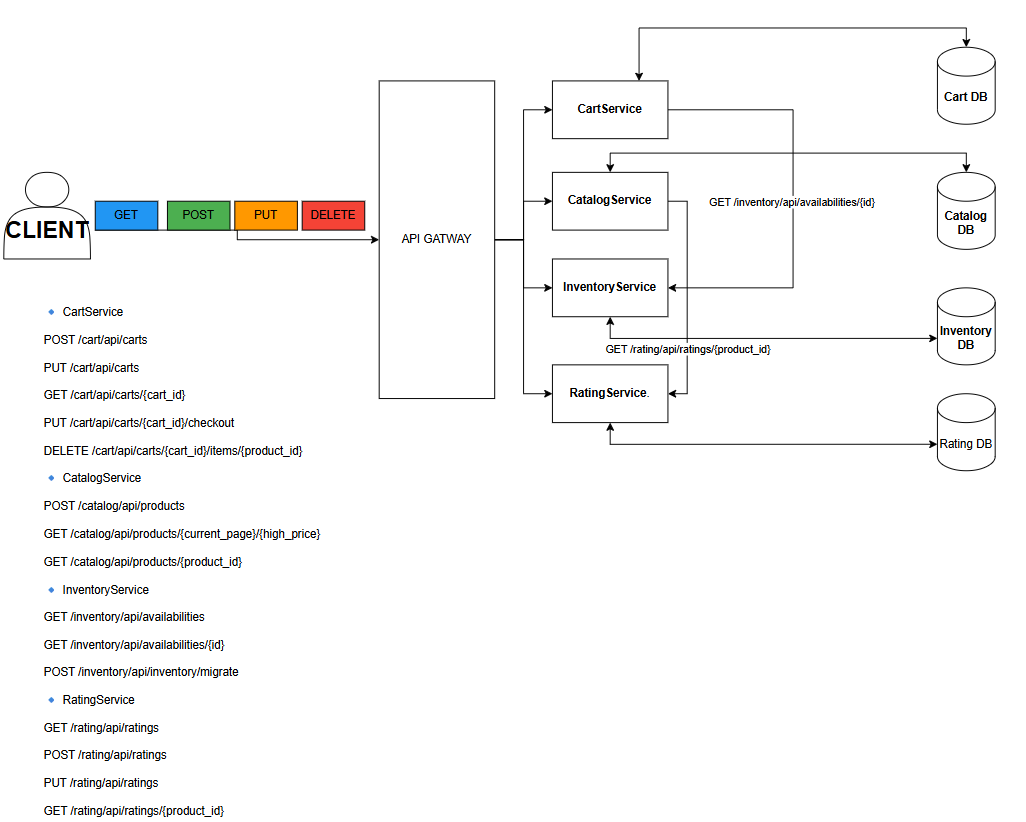
## Bài 1. Vẽ lại sơ đồ kiến trúc SPA (Single Page Application):



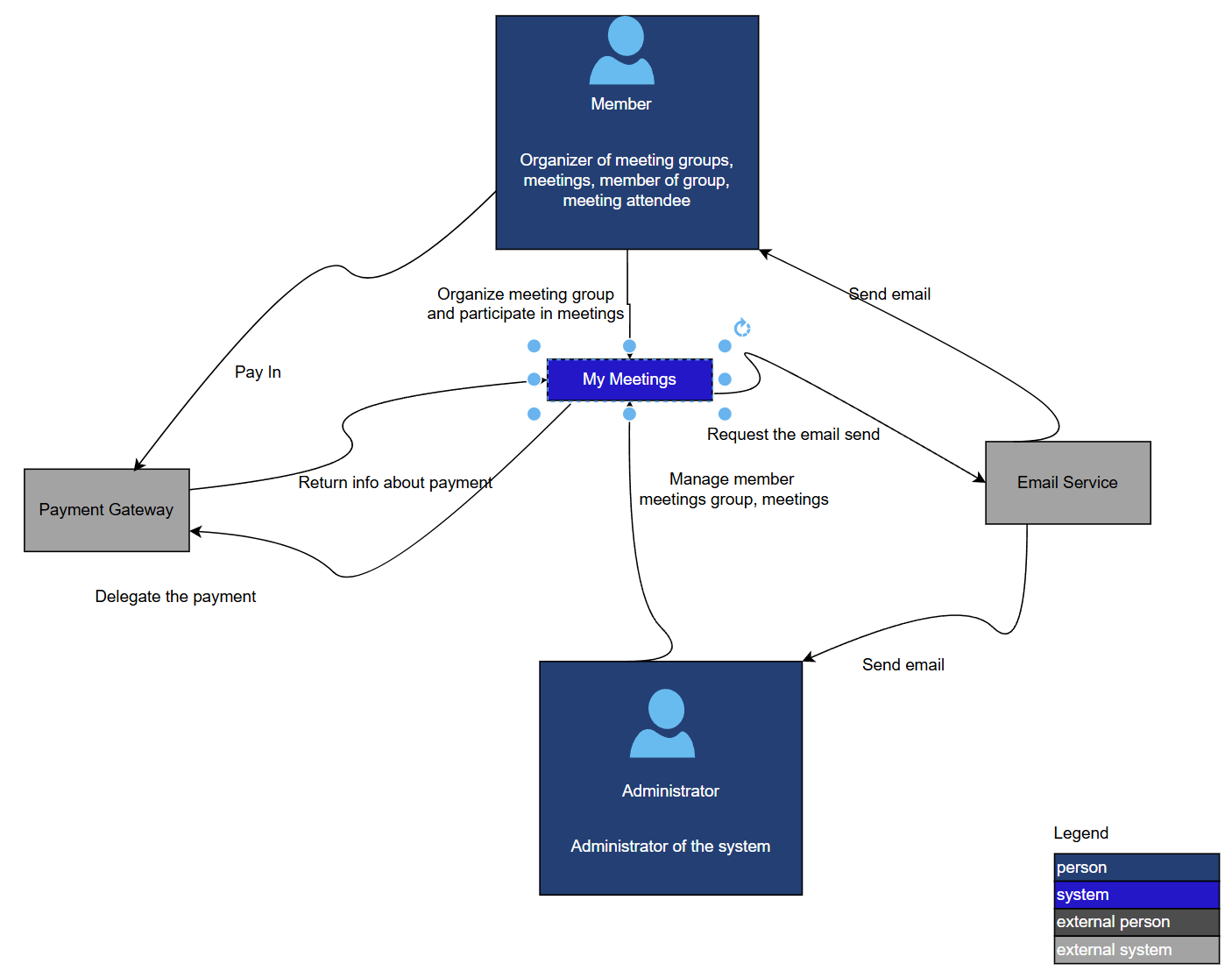
## Bài 2. Vẽ sơ đồ triển khai CI/CD:

****

## Bài 3. Vẽ sơ đồ API của hệ thống:



## Bài 4. Vẽ sơ đồ C1 - System Context và giải thích:



**Giải thích :**

1. **Ý nghĩa sơ đồ**

Sơ đồ C1 (System Context Diagram) mô tả **hệ thống “My Meetings”** ở mức khái quát, cho thấy:

* Hệ thống nằm ở trung tâm (ô màu xanh đậm).
* Các **tác nhân bên ngoài (Actor)** (người dùng, quản trị viên) và **hệ thống bên ngoài** (cổng thanh toán, dịch vụ email) có quan hệ trực tiếp với hệ thống.
* Các **luồng thông tin/chức năng chính** trao đổi giữa hệ thống và các tác nhân, hệ thống bên ngoài.

1. **Các thành phần trong sơ đồ**

**Member (Người dùng – thành viên)**

* Vai trò: Người tổ chức nhóm họp, tham gia họp, thành viên trong nhóm.
* Chức năng:
* Tổ chức nhóm họp, tham gia họp thông qua hệ thống **My Meetings**.
* Thực hiện thanh toán phí qua **Payment Gateway**.
* Nhận email thông báo từ hệ thống qua **Email Service**.

**Administrator (Quản trị viên hệ thống)**

* Vai trò: Quản lý và vận hành hệ thống.
* Chức năng:
  + Quản lý thành viên, nhóm họp, các cuộc họp.
  + Thông qua hệ thống **My Meetings** để gửi yêu cầu email đến **Email Service**.

**My Meetings (Hệ thống trung tâm)**

* Là hệ thống chính mà ta xây dựng.
* Đảm nhận:
  + Tổ chức và quản lý các cuộc họp, nhóm họp, thành viên.
  + Xử lý việc thanh toán (kết nối với Payment Gateway).
  + Kích hoạt và gửi yêu cầu dịch vụ email.
  + Cho phép quản trị viên giám sát, quản lý.

**Payment Gateway (Cổng thanh toán)**

* Hệ thống bên ngoài chuyên xử lý giao dịch tài chính.
* Nhiệm vụ:
  + Nhận yêu cầu thanh toán từ thành viên.
  + Trả về thông tin kết quả giao dịch cho **My Meetings**.

**Email Service (Dịch vụ Email)**

* Hệ thống bên ngoài chuyên xử lý việc gửi email.
* Nhiệm vụ:
* Nhận yêu cầu gửi email từ **My Meetings**.
* Gửi email thông báo đến Member hoặc Administrator.

1. **Luồng tương tác chính**

**Member ↔ My Meetings**

* Member sử dụng hệ thống để tạo/ tham gia nhóm họp.
* Khi cần thanh toán, My Meetings sẽ điều hướng Member đến Payment Gateway.

**Member ↔ Payment Gateway ↔ My Meetings**

* Member trả phí qua Payment Gateway.
* Payment Gateway trả về thông tin xác nhận cho My Meetings.

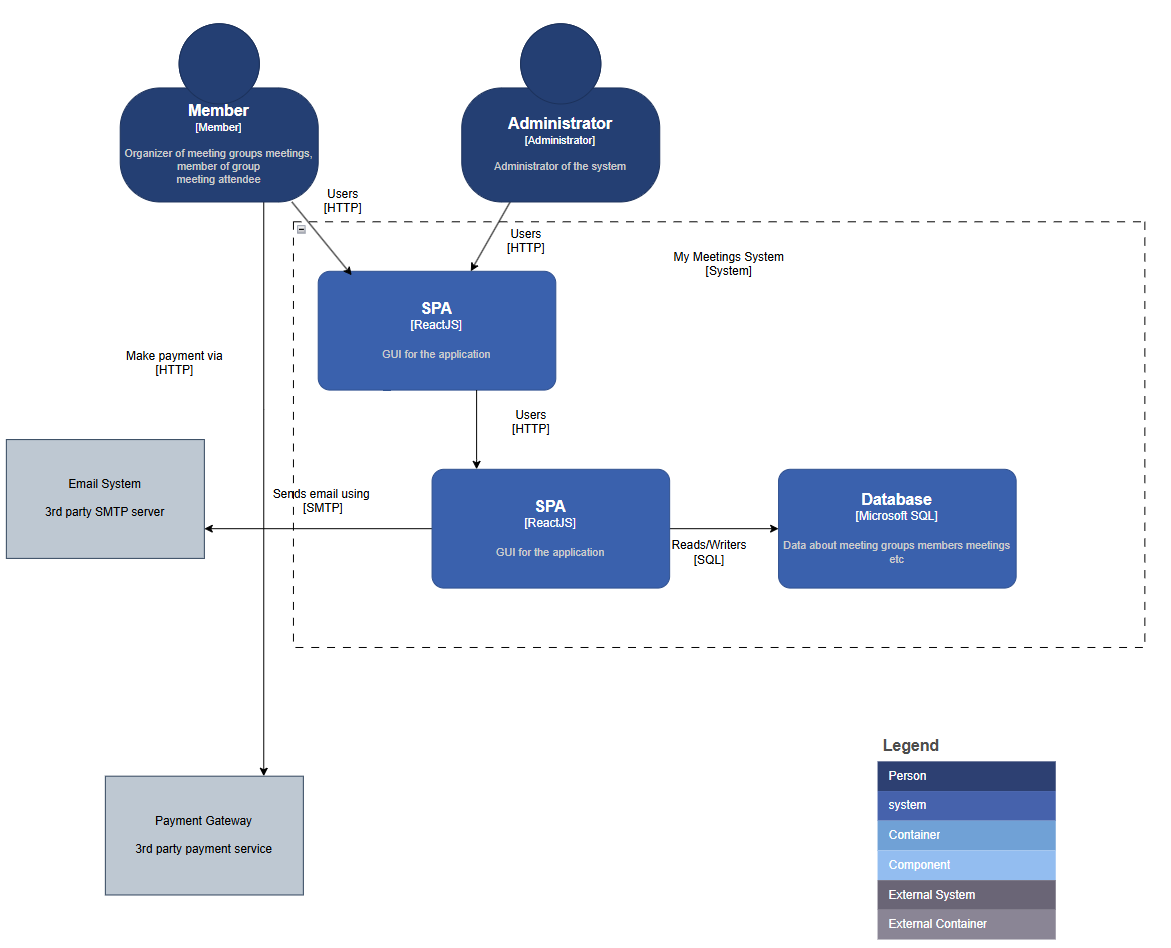
**My Meetings ↔ Email Service ↔ Member / Administrator**

* Hệ thống My Meetings gửi yêu cầu đến Email Service.
* Email Service gửi email đến Member hoặc Administrator.

**Administrator ↔ My Meetings**

* Administrator quản lý thành viên, nhóm họp và các cuộc họp thông qua hệ thống.

## Bài 5. Vẽ sơ đồ C2 – Container và giải thích:



**Giải thích**

**Tổng quan**

Đây là sơ đồ **C2 – Container**, thể hiện cách hệ thống **My Meetings System** được chia thành các **containers** (ứng dụng web, cơ sở dữ liệu, các dịch vụ bên ngoài) và cách chúng giao tiếp với nhau. Nó ở mức cao hơn so với component (C3), tập trung vào **các khối chính** thay vì chi tiết bên trong.

**Các thành phần chính**

**Người dùng**

* **Member**: người dùng bình thường, có thể tổ chức nhóm họp, tham gia cuộc họp, đăng ký thành viên.
* **Administrator**: quản trị viên, có quyền quản lý hệ thống.

🡪Cả hai đều tương tác với hệ thống thông qua **SPA**.

**Containers trong hệ thống**

**SPA (ReactJS)**

* Là giao diện người dùng (frontend) cho toàn bộ hệ thống.
* Người dùng (Member và Administrator) thao tác qua SPA bằng HTTP.
* SPA giao tiếp với backend để xử lý dữ liệu và hiển thị nội dung.
* Có 2 SPA thể hiện (có thể là SPA chính và một SPA phụ cho quản trị hoặc chức năng riêng).

**Database (Microsoft SQL)**

* Lưu trữ dữ liệu về nhóm họp, thành viên, các cuộc họp…
* SPA sẽ đọc/ghi dữ liệu thông qua SQL.

**Hệ thống bên ngoài**

**Email System (3rd party SMTP server)**

* Được sử dụng để gửi email (ví dụ: email xác nhận, thông báo lịch họp…).
* SPA gửi yêu cầu gửi email thông qua giao thức SMTP.

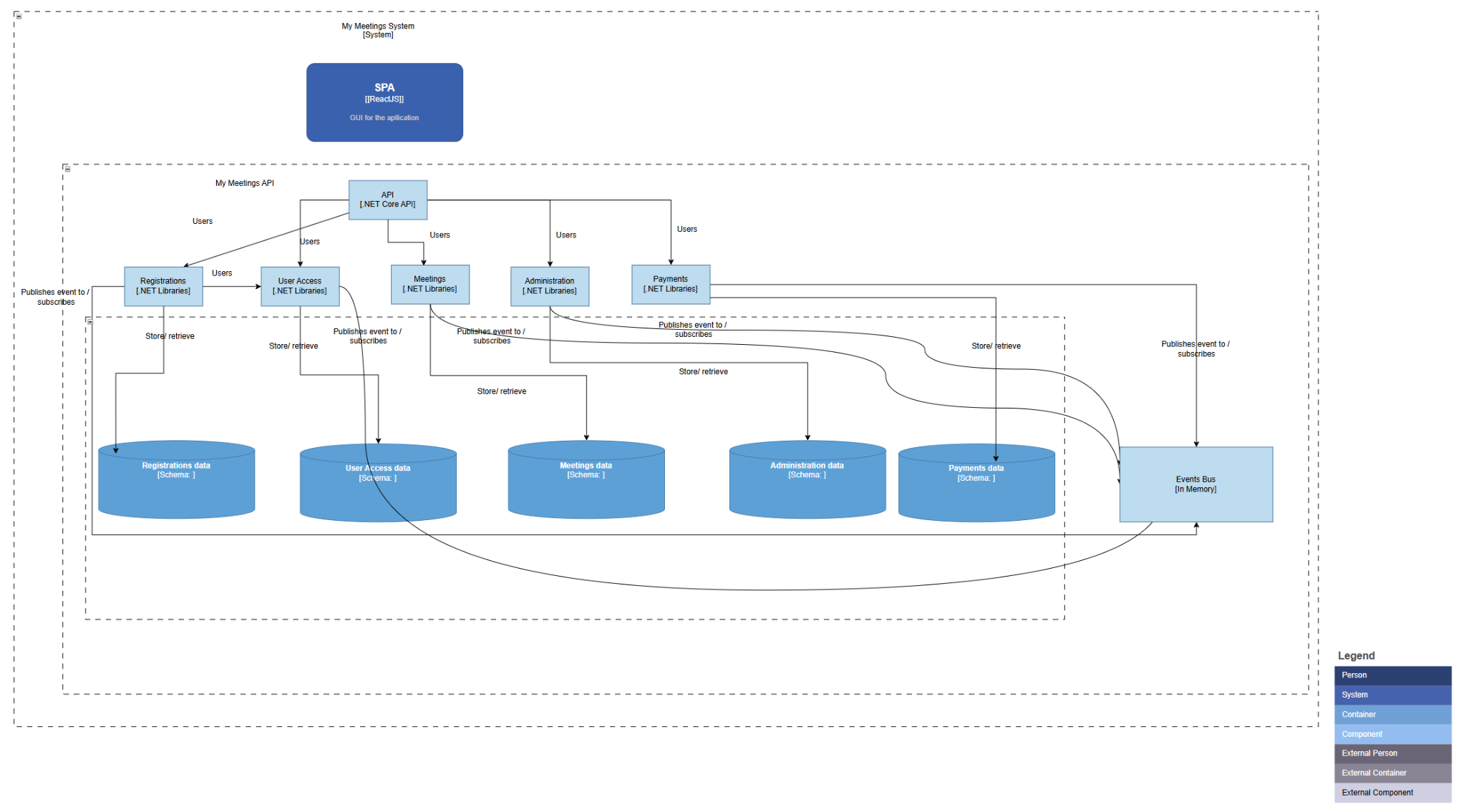
**Payment Gateway (3rd party payment service)**

* Được sử dụng để xử lý thanh toán cho hệ thống.
* Người dùng thực hiện thanh toán qua HTTP.
* Kết hợp với Email System để gửi hóa đơn hoặc xác nhận giao dịch.

**Luồng hoạt động chính**

1. **Người dùng (Member/Administrator)** đăng nhập và sử dụng SPA để quản lý hoặc tham gia các cuộc họp.
2. **SPA** gửi/nhận dữ liệu từ **Database (Microsoft SQL)** để lưu thông tin về người dùng, nhóm, cuộc họp.
3. Khi cần gửi thông báo, **SPA** gọi **Email System (SMTP)** để gửi email cho người dùng.
4. Khi có thanh toán, **SPA** kết nối tới **Payment Gateway** (dịch vụ thanh toán bên thứ ba).

## Bài 6. Vẽ sơ đồ C3 – Component (High-Level) và giải thích:



**Giải thích**

**Tổng quan**

Sơ đồ mô tả kiến trúc **My Meetings System** ở mức component. Nó cho thấy cách ứng dụng được chia thành các phần chính (frontend, backend API, các module .NET libraries, database riêng và event bus) và cách các phần này giao tiếp với nhau.

**Các thành phần chính**

1. **SPA (ReactJS)**

* Là ứng dụng giao diện người dùng (Single Page Application).
* Người dùng thao tác trên SPA để thực hiện các chức năng như đăng ký, tạo cuộc họp, thanh toán...
* SPA giao tiếp với API backend để xử lý dữ liệu.

1. **My Meetings API (.NET Core API)**

* Đây là backend chính, đóng vai trò làm cầu nối giữa SPA và dữ liệu hệ thống.
* Được chia thành nhiều component dưới dạng .NET Libraries, mỗi component phụ trách một chức năng riêng:
* **Registrations**: quản lý thông tin đăng ký.
* **User Access**: quản lý quyền và truy cập người dùng.
* **Meetings**: quản lý các cuộc họp.
* **Administration**: quản trị hệ thống.
* **Payments**: xử lý thanh toán.
* Mỗi component có thể:
* Giao tiếp với database riêng.
* Phát sinh sự kiện (publish event) lên Event Bus để các component khác có thể phản hồi.

1. **Databases (Schemas)**

* Mỗi component có một database/schema riêng:
* Registrations data
* User Access data
* Meetings data
* Administration data
* Payments data
* Cách thiết kế này tách biệt rõ từng domain, giúp hệ thống dễ bảo trì và mở rộng theo hướng microservices.

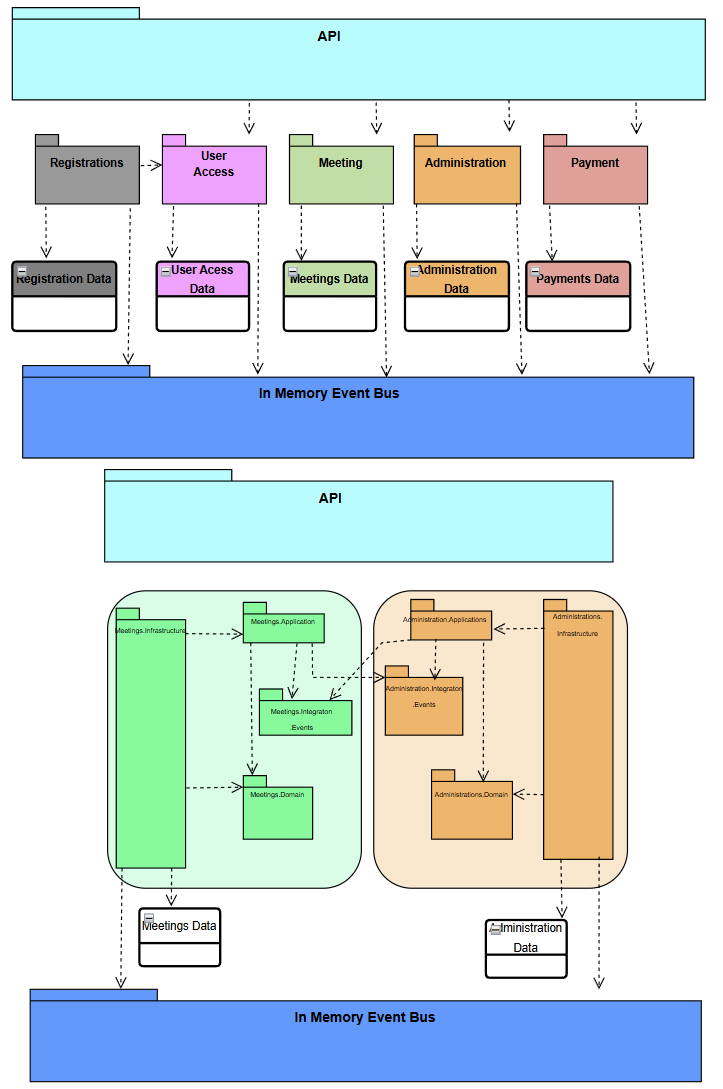
1. **Event Bus (In Memory)**

* Là cơ chế truyền thông điệp nội bộ.
* Các component publish sự kiện (ví dụ: người dùng đã đăng ký, thanh toán hoàn tất).
* Những component khác có thể subscribe để xử lý tiếp theo.
* Cơ chế này giúp các phần trong hệ thống ít phụ thuộc lẫn nhau và hỗ trợ kiến trúc hướng sự kiện.

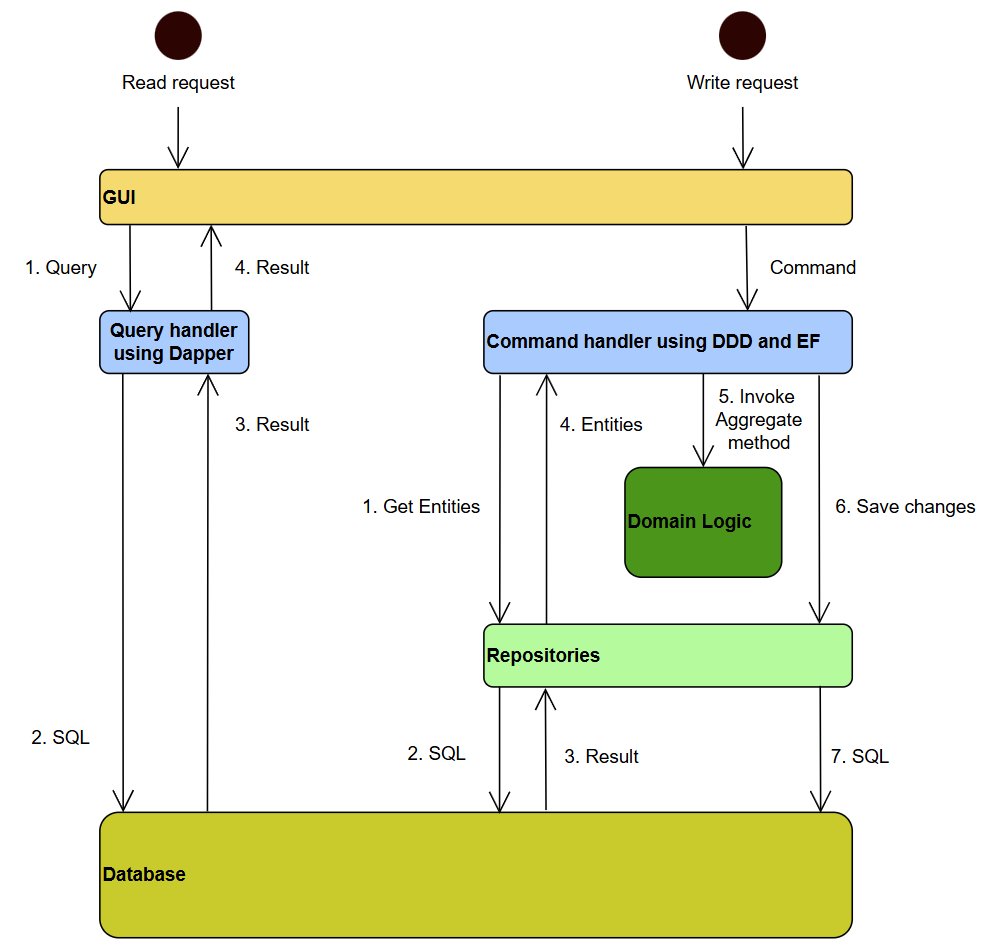
**Luồng hoạt động**

* Người dùng thao tác trên SPA (ReactJS).
* SPA gửi request tới My Meetings API.
* API định tuyến request đến component tương ứng (Registrations, Meetings, Payments...).
* Component lưu/đọc dữ liệu trong database riêng.
* Nếu có sự kiện cần chia sẻ, component publish lên Event Bus.
* Các component khác có thể nhận sự kiện từ Event Bus để xử lý tiế

## Bài 7. Vẽ sơ đồ C3 – Component (Module-Level) sau:



## Bài 8. Vẽ sơ đồ xử lý 1 request:



**VÍ DỤ MINH HỌA (BEST PRACTICES)**

**Tham khảo**:

https://github.com/kgrzybek/modular-monolith-with-ddd <https://vietnam-devs.github.io/coolstore-microservices>

**Tổng quan kiến trúc**

Các hình trên mô tả hệ thống (theo tinh thần C4) dự án *Modular Monolith with DDD* của Kamil Grzybek.

* **C1**: Hệ thống Meetup-like với users/admin/payer, quy trình đề xuất-duyệt nhóm, cuộc họp, thanh

toán.

* **C2**: Monolith theo module, **API mỏng + Event Bus trong bộ nhớ**, module sở hữu dữ liệu riêng.
* **C3**: Mỗi module tách **Application/Domain/Infrastructure/IntegrationEvents**, xử lý CQRS +

events.

* **C4**: Code-level có ví dụ aggregate cho **Meeting Group** (được đính kèm sơ đồ trong README).

**Mô tả chi tiết:**

**C1 — System Context (tóm lược)**

* Hệ thống là một **Modular Monolith** phục vụ miền “Meeting Groups/Meetup”: người dùng đăng ký tài khoản, đề xuất/duyệt nhóm họp, tạo cuộc họp, tham gia, bình luận; có thanh toán phí đăng ký/fee sự kiện.
* Các actor/chức năng cấp cao được hiện thực qua các **module** sau: **User Access**, **Registrations**, **Meetings**, **Administration**, **Payments**; API mỏng ở phía trước, và một **In-Memory Events Bus** dùng để tích hợp bất đồng bộ giữa các module.

**C2 — Container View**

* **API (ASP.NET Core REST)**

Vai trò: nhận request → xác thực/ủy quyền (dựa vào User Access) → gửi **Command/Query** tới module đích → trả response. Không chứa business logic. API nói chuyện với module qua giao diện nhỏ (ports) do từng module cung cấp.

* **Các Module nghiệp vụ** (nằm trong cùng tiến trình — monolith, nhưng đóng gói chặt chẽ và độc lập dữ liệu):
  + **User Access**: xác thực/ủy quyền.
  + **Registrations**: đăng ký người dùng.
  + **Meetings**: bounded context tạo nhóm họp/cuộc họp, quản lý tham dự, bình luận…
  + **Administration**: duyệt/khước từ **Meeting Group Proposal** và các tác vụ quản trị.
  + **Payments**: thanh toán subscription, fee cuộc họp, gia hạn.
  + **In-Memory Events Bus**: cơ chế Pub/Sub để **các module chỉ tích hợp bất đồng bộ bằng sự kiện** (cấm gọi trực tiếp qua method).
* **Các giả định/luật tích hợp quan trọng**
  + API không có logic ứng dụng;
  + API ↔ Module qua interface nhỏ (Command/Query);
  + Module-to-Module chỉ qua Event Bus;
  + **Mỗi module sở hữu dữ liệu riêng trong schema riêng** (không chia sẻ), và có thể tách ra DB riêng khi cần;
  + Chỉ được phụ thuộc vào **assemblies IntegrationEvents** của module khác;
  + Mỗi module có Composition Root/IoC riêng và được API khởi tạo;
  + Mức đóng gói cao (public tối thiểu).

**C3 — Component View (mẫu cấu trúc bên trong từng module)**

Mỗi **module** tuân theo **Clean Architecture** và chia thành các *submodules/assemblies* chuẩn:

* **Application**: xử lý use case (Command/Query), domain events, integration events, internal

commands.

* **Domain**: mô hình domain (DDD) của **Bounded Context** tương ứng.
* **Infrastructure**: khởi tạo module, nền tảng chạy nền, truy cập dữ liệu, giao tiếp Event Bus & hệ thống

ngoài.

* **IntegrationEvents**: hợp đồng sự kiện công bố ra Event Bus (đây là **điểm duy nhất** cho module khác

phụ thuộc).

**Nguyên tắc domain** áp dụng: encapsulation cao (mặc định private/internal), persistence-ignorant (POCO, không phụ thuộc hạ tầng), domain giàu hành vi, tránh “primitive obsession” bằng Value Object, đặt tên theo ngôn ngữ nghiệp vụ, dễ kiểm thử.

**Dòng chảy xử lý (Request & CQRS)**

1. **API** nhận HTTP request, xác thực/ủy quyền bằng **User Access** →
2. API gửi **Command/Query** tới **Application** của module đích →
3. Application gọi **Domain** để thực thi nghiệp vụ, phát **Domain Event** nội bộ nếu có →
4. Khi cần tích hợp liên-module, module **publish Integration Event** lên **In-Memory Events Bus** để

module khác **subscribe** và phản ứng bất đồng bộ. (Phong cách EDA trong một monolith.)

**Dữ liệu & biên giới module**

* **Data ownership**: mỗi module quản lý **schema dữ liệu riêng**, không chia sẻ bảng; có thể nâng cấp thành DB riêng nếu cần.
* **Giao tiếp dữ liệu** giữa module: qua **Integration Events** (không đọc chéo DB, không gọi method

chéo).

**Bảo mật & khởi tạo**

* **User Access** phụ trách authN/authZ; API dựa vào đây trước khi ủy quyền use case vào module.
* **Khởi tạo module**: API gọi **Initialize(...)** tĩnh của từng module trong Startup, truyền cấu hình (connection string, logger, email config…), dựng Composition Root, khởi động **Quartz** (jobs) và **Events Bus** cho module.

**Kiểm thử & chất lượng (khung hỗ trợ trong repo)**

Repo minh họa: **unit test domain**, **integration test**, **system integration testing**, **event sourcing**, **database change management**, **CI**, **static analysis**, **mutation testing**—dùng để đảm bảo kiến trúc & hiện thực ở mức production-ready.

# BÀI TẬP ỨNG DỤNG

**HỆ THỐNG QUẢN LÝ THƯ VIỆN TRỰC TUYẾN**

**Business Context**

* **Mục tiêu kinh doanh**: Giúp thư viện hiện đại hóa việc mượn/trả sách, giảm phụ thuộc vào thủ thư, tăng trải nghiệm người dùng, và quản lý kho sách hiệu quả.
* **Các tác nhân chính**:
  + **Người đọc**: Tìm kiếm sách, mượn sách online, gia hạn, xem lịch sử mượn.
  + **Thủ thư/Admin**: Quản lý kho sách, xử lý yêu cầu mượn/trả, theo dõi phí trễ hạn.
  + **Hệ thống thanh toán**: Xử lý phí thành viên hoặc phí trễ hạn.
* **Ràng buộc & giá trị**: Hệ thống phải ổn định, bảo mật dữ liệu người đọc, cho phép nhiều người dùng

đồng thời.

Business Context = **Mục tiêu kinh doanh + Các tác nhân + Giá trị mang lại + Ràng buộc chính**

**Thiết kế kiến trúc phần mềm theo chuẩn C4 model** theo 4 mức của C4:

* **Level 1 (Context Diagram)**: Hệ thống và các tác nhân bên ngoài.
* **Level 2 (Container Diagram)**: Các container (ứng dụng web, mobile app, database, service...).
* **Level 3 (Component Diagram)**: Các thành phần trong một container quan trọng.
* **Level 4 (Code/Implementation Diagram)**: Chi tiết lớp hoặc cấu trúc code.

**Yêu cầu**:

* **Mô tả**: Xây dựng hệ thống cho phép người dùng mượn/trả sách, quản lý kho sách, và thanh toán phí trễ hạn.
* **Yêu cầu**:
  + Vẽ sơ đồ **Context**: Thư viện online, Người dùng, Thủ thư, Hệ thống thanh toán.
  + Vẽ sơ đồ **Container**: Web App, Mobile App, Database, Payment Service.
  + Vẽ sơ đồ **Component**: Bên trong Web App, có module tìm kiếm, module mượn/trả, module quản lý tài khoản.
  + Vẽ sơ đồ **Code**: Ví dụ class diagram cho module “Quản lý mượn sách”.
* Bạn có thể vẽ bằng **PlantUML, Structurizr DSL, hoặc draw.io** để thể hiện trực quan.
* Khi làm bài tập, hãy mô tả **các quyết định thiết kế chính** (ví dụ: tại sao dùng SPA, tại sao tách

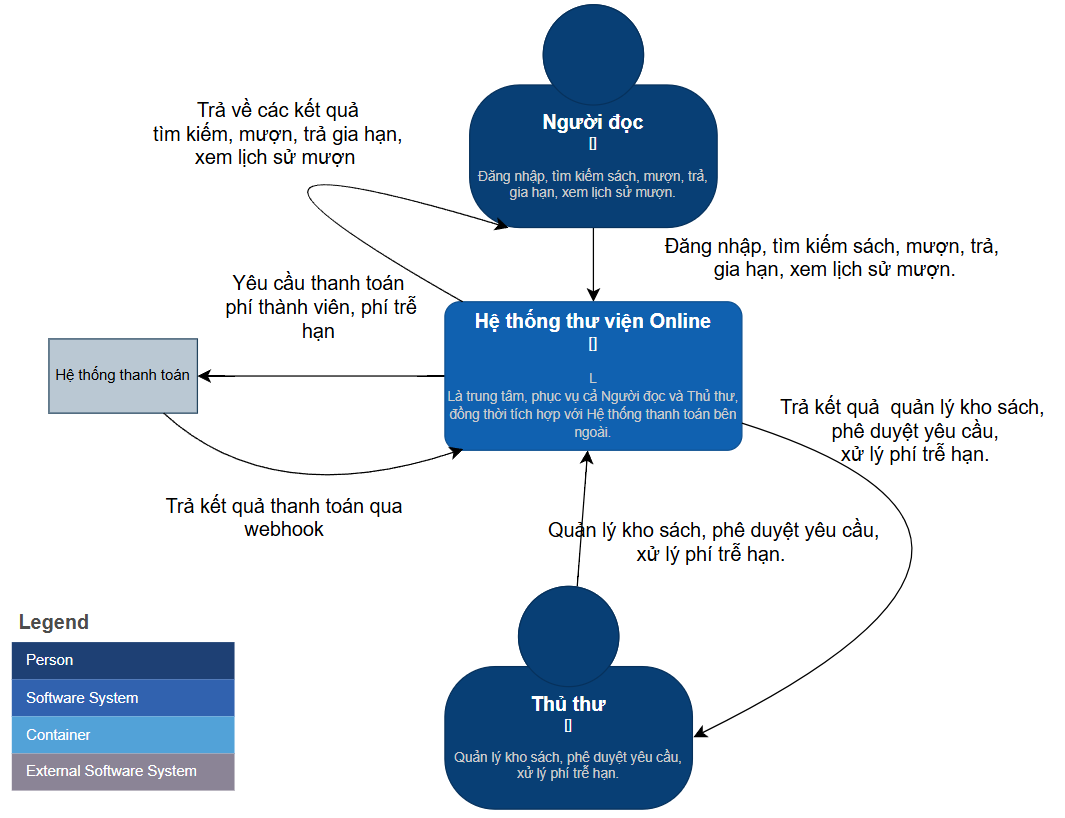
mobile/web…).

**Bài làm**

## Tổng quan kiến trúc

* **C1 — System Context:** Hệ thống thư viện trực tuyến với các tác nhân chính là Người đọc, Thủ thư/Admin, và Hệ thống thanh toán. Người đọc có thể tìm kiếm, mượn, gia hạn, trả sách và theo dõi lịch sử; Thủ thư quản lý kho và xử lý phí; Hệ thống thanh toán đảm nhận phí thành viên và phí trễ hạn.
* **C2 — Container View:** Kiến trúc gồm Web App (SPA) và Mobile App cho người dùng; Backend API xử lý nghiệp vụ (API mỏng, CQRS); Database PostgreSQL + Redis để quản lý dữ liệu và cache; Search Engine (Elasticsearch) cho tìm kiếm nâng cao; Payment Service tích hợp cổng thanh toán ngoài.
* **C3 — Component View:** Backend API được chia thành các module: Auth/Identity (xác thực/ủy quyền), Catalog (quản lý sách), Borrowing (mượn/trả/gia hạn), Accounts (thông tin người đọc, lịch sử mượn), Payments Integration (xử lý giao dịch, webhook), Notifications (thông báo), và Infrastructure (ORM, outbox, adapters). Hỗ trợ xử lý CQRS và phát/nhận sự kiện.
* **C4 — Code/Implementation:** Ở mức chi tiết code, ví dụ module Quản lý mượn sách có aggregate chính: Member, BookCopy, Loan, Fine. Các domain events như BookBorrowed, BookReturned, LoanExtended đảm bảo dòng chảy nghiệp vụ mượn/trả/gia hạn sách.

## Level 1 (Context Diagram): Hệ thống và các tác nhân bên ngoài.



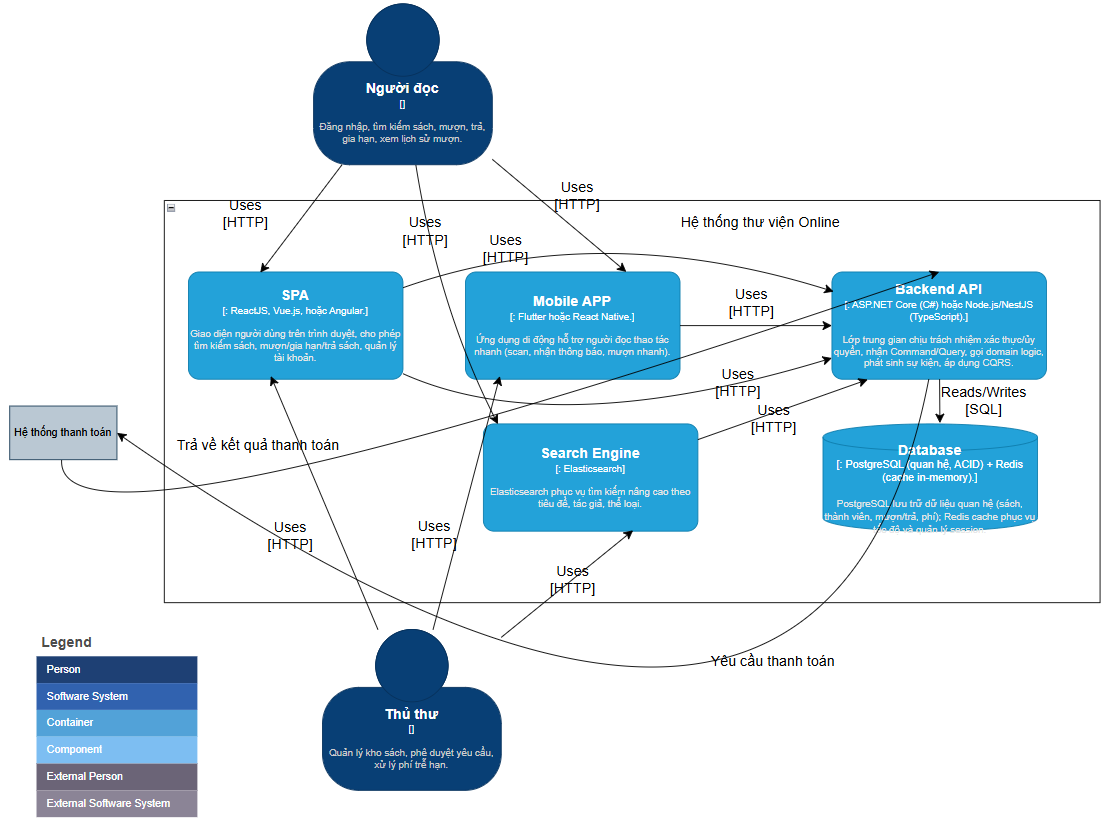
**Mô tả chi tiết:**

**C1 — System Context**

Ở mức độ **Context**, hệ thống được nhìn từ góc độ toàn cục, tập trung vào các tác nhân bên ngoài và cách họ tương tác với hệ thống:

* **Người đọc (Reader)**: đăng nhập, tìm kiếm sách, mượn, trả, gia hạn, xem lịch sử mượn.
* **Thủ thư/Admin (Librarian/Admin)**: quản lý kho sách, phê duyệt yêu cầu, xử lý phí trễ hạn.
* **Hệ thống thanh toán (Payment Provider)**: thu phí thành viên, phí trễ hạn, trả kết quả thanh toán qua webhook.
* **Library Online System** là trung tâm, phục vụ cả Người đọc và Thủ thư, đồng thời tích hợp với Hệ thống thanh toán bên ngoài.

## Level 2 (Container Diagram): Các container (ứng dụng web, mobile app, database, service...).



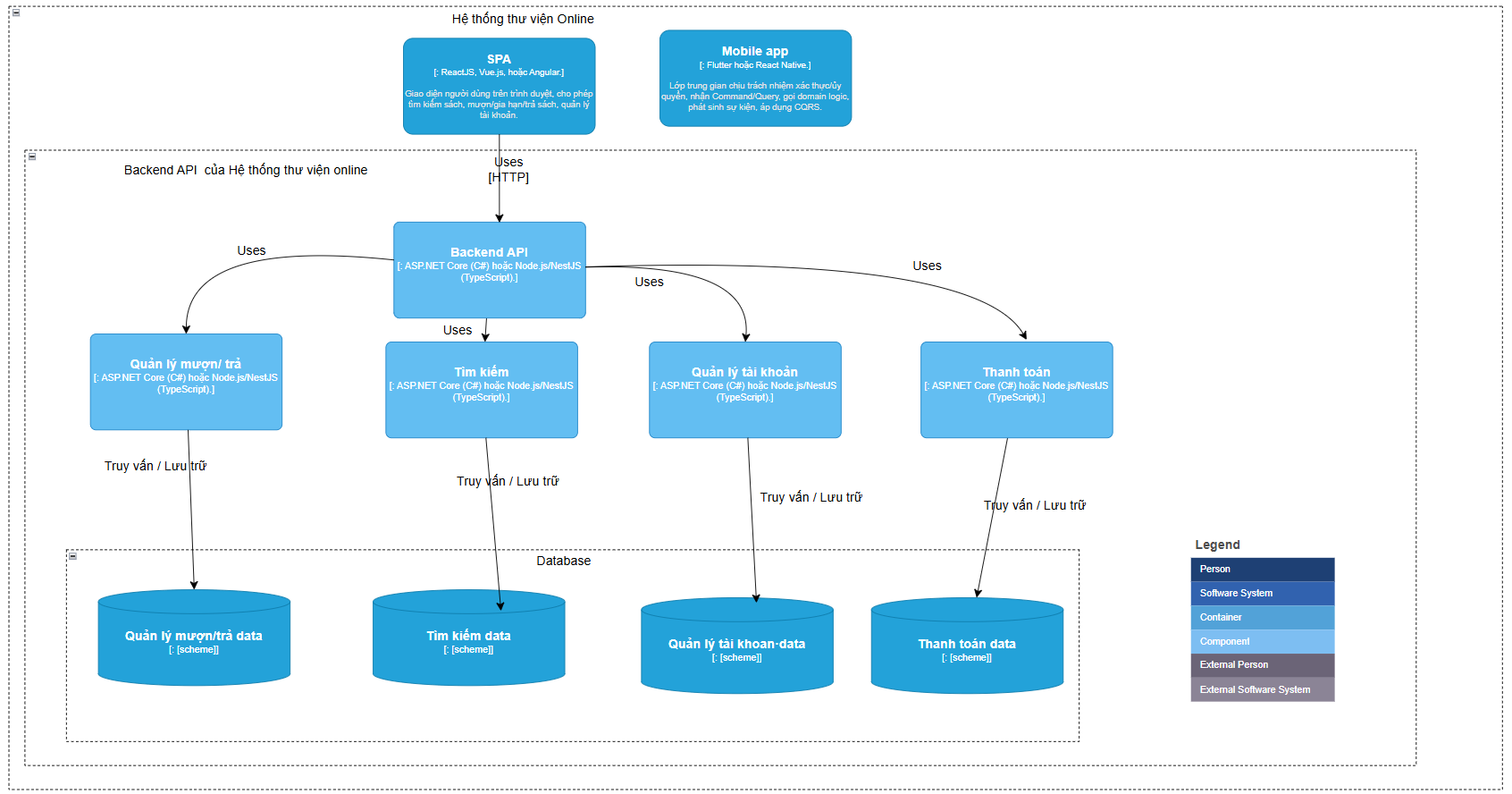
**Mô tả chi tiết:**

**C2 — Container View**

Ở mức **Container**, hệ thống được chia thành các thành phần triển khai chính:

* **Web App (SPA)**: giao diện người dùng trên trình duyệt, cho phép tìm kiếm sách, mượn/gia hạn/trả sách, quản lý tài khoản.
* **Mobile App**: ứng dụng di động hỗ trợ người đọc thao tác nhanh (scan, nhận thông báo, mượn nhanh).
* **Backend API**: lớp trung gian chịu trách nhiệm xác thực/ủy quyền, nhận Command/Query, gọi domain logic, phát sinh sự kiện, áp dụng CQRS.
* **Database**: PostgreSQL lưu trữ dữ liệu quan hệ (sách, thành viên, mượn/trả, phí); Redis cache phục vụ tốc độ và quản lý session.
* **Search Engine**: Elasticsearch phục vụ tìm kiếm nâng cao theo tiêu đề, tác giả, thể loại.
* **Payment Service**: hệ thống thanh toán ngoài (VNPay/MoMo/Stripe) để xử lý giao dịch.

## Level 3 (Component Diagram): Các thành phần trong một container quan trọng.



**Mô tả chi tiết:**

**C3 — Component View (High-level Web App Modules)**

Web App (SPA) với 4 module chính.

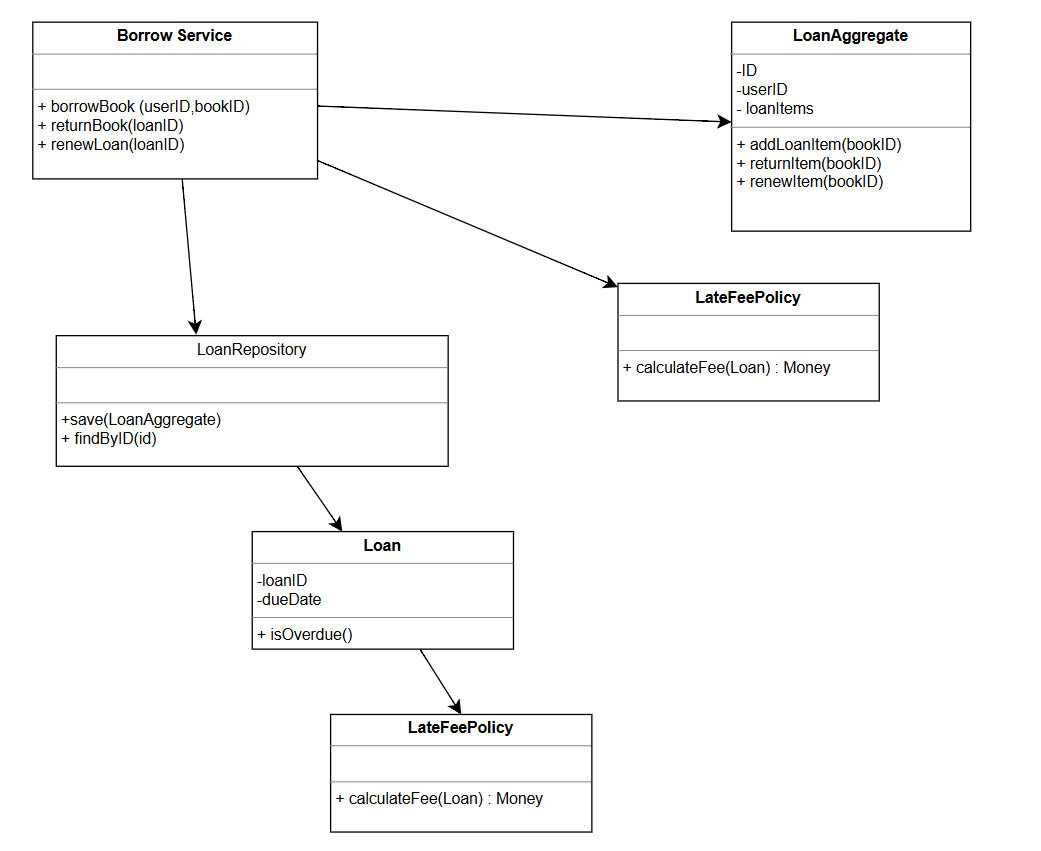
**Module descriptions**

* **Search Module — *Tìm kiếm danh mục sách***
  + Nhiệm vụ chính: nhận input tìm kiếm, gọi API GET /books/search, hiển thị chi tiết sách.
  + Lưu ý: UI/state client‑side, không chứa business logic domain.
* **Borrow/Return Module — *Luồng mượn, trả, gia hạn***
  + Nhiệm vụ chính: POST /loans/borrow, POST /loans/return, POST /loans/extend, GET /loans/active.
  + Lưu ý: xác thực bằng JWT, hiển thị due date, trạng thái Active/Overdue.
* **Account Management Module — *Quản lý tài khoản người đọc***
  + Nhiệm vụ chính: GET /members/me, GET /members/me/loans, GET /members/me/fines, PATCH /members/me.
  + Lưu ý: hiển thị hồ sơ, lịch sử, cập nhật thông tin.
* **Payments Module — *Thanh toán phí thành viên và phí trễ hạn***
  + Nhiệm vụ chính: POST /payments/create, GET /payments/{id}, GET /payments/history.
  + Lưu ý: hiển thị biên lai, trạng thái giao dịch, thông báo kết quả.

**Key assumptions**

1. Web App không chứa business logic domain; chỉ UI và gọi API.
2. Web App ↔ Backend API qua REST/JSON.
3. Mỗi module UI có biên giới rõ ràng (route, state, component boundaries).
4. Auth bắt buộc với mượn/trả/gia hạn/thanh toán; guard theo RBAC.
5. Không truy cập DB trực tiếp; dữ liệu chỉ từ API.
6. Client có logging lỗi, fallback khi request lỗi.

## Level 4 (Code/Implementation Diagram): Chi tiết lớp hoặc cấu trúc code.



**Mô tả chi tiết:**

**Các lớp chính trong hệ thống**

**1. Borrow Service**

* **Vai trò**: lớp dịch vụ chính để xử lý các thao tác mượn, trả và gia hạn sách.
* **Phương thức**:
  + borrowBook(userID, bookID) – cho phép người dùng mượn sách.
  + returnBook(loanID) – trả sách đã mượn.
  + renewLoan(loanID) – gia hạn thêm thời gian mượn sách.

**2. LoanAggregate**

* **Vai trò**: đối tượng tổng hợp, quản lý thông tin toàn bộ các khoản mượn của một người dùng.
* **Thuộc tính**:
  + ID – mã định danh của LoanAggregate.
  + userID – ID của người dùng.
  + loanItems – danh sách các sách đã mượn.
* **Phương thức**:
  + addLoanItem(bookID) – thêm sách mới vào danh sách mượn.
  + returnItem(bookID) – trả một quyển sách.
  + renewItem(bookID) – gia hạn một quyển sách.

**3. LoanRepository**

* **Vai trò**: quản lý việc lưu trữ và truy xuất dữ liệu về LoanAggregate.
* **Phương thức**:
  + save(LoanAggregate) – lưu thông tin khoản mượn.
  + findByID(id) – tìm một khoản mượn theo ID.

**4. Loan**

* **Vai trò**: thể hiện một khoản mượn cụ thể.
* **Thuộc tính**:
  + loanID – ID của khoản mượn.
  + dueDate – hạn trả sách.
* **Phương thức**:
  + isOverdue() – kiểm tra xem khoản mượn đã quá hạn hay chưa.

**5. LateFeePolicy**

* **Vai trò**: chính sách tính phí phạt khi mượn sách quá hạn.
* **Phương thức**:
  + calculateFee(Loan): Money – tính số tiền phạt dựa trên tình trạng quá hạn của khoản mượn.

**Quan hệ giữa các lớp**

* **Borrow Service**: gọi đến LoanAggregate để thêm, trả hoặc gia hạn sách.
* **Borrow Service** cũng tương tác với LateFeePolicy để tính phí phạt.
* **LoanRepository**: là nơi Borrow Service lưu hoặc tìm kiếm thông tin các khoản mượn (LoanAggregate).
* **LoanAggregate** quản lý nhiều **Loan** (các khoản mượn đơn lẻ).
* **Loan** có quan hệ với **LateFeePolicy** để kiểm tra phí phạt nếu quá hạn.

**Ý nghĩa**

Biểu đồ này mô tả luồng nghiệp vụ **mượn – trả – gia hạn sách** trong hệ thống thư viện.

* **Borrow Service** đóng vai trò trung tâm, xử lý yêu cầu từ người dùng.
* **LoanAggregate** và **Loan** quản lý dữ liệu mượn sách.
* **LoanRepository** đóng vai trò lưu trữ.
* **LateFeePolicy** xử lý logic liên quan đến phí phạt.

⎯⎯⎯ Hết ⎯⎯