**Lab 3**

**Thành viên nhóm**

Đỗ Nguyễn Nhật Trường – 3122411230

Nguyễn Thị Diệu Linh - 3122411112

Nguyễn Tấn Thành - 3122411192

Nguyễn Thanh Tịnh – 312241121

Mục lục

[**I.** **BÀI TẬP MINH HỌA** 1](#_Toc209608789)

[**Bài 1.** **Vẽ lại sơ đồ kiến trúc SPA (Single Page Application).** 1](#_Toc209608790)

[**Bài 2.** **Vẽ sơ đồ triển khai CI/CD.** 2](#_Toc209608791)

[**Bài 3.** **Vẽ sơ đồ API của hệ thống.** 3](#_Toc209608792)

[**Bài 4.** **Vẽ sơ đồ C1 - System Context và giải thích.** 4](#_Toc209608793)

[**Bài 5.** **Vẽ sơ đồ C2 – Container và giải thích:** 6](#_Toc209608794)

[**Bài 6.** **Vẽ sơ đồ C3 – Component (High-Level) và giải thích:** 8](#_Toc209608795)

[**Bài 7.** **Vẽ sơ đồ C3 – Component (Module-Level) sau:** 10](#_Toc209608796)

[**Bài 8.** **Vẽ sơ đồ xử lý 1 request:** 11](#_Toc209608797)

[**II.** **BÀI TẬP ỨNG DỤNG** 12](#_Toc209608798)

[i. **Mô tả:** Xây dựng hệ thống cho phép người dùng mượn/trả sách, quản lý kho sách, và thanh toán phí trễ hạn. 12](#_Toc209608799)

[**ii.** **Yêu cầu:** 18](#_Toc209608800)

[**Vẽ sơ đồ Context:** Thư viện online, Người dùng, Thủ thư, Hệ thống thanh toán. 18](#_Toc209608801)

[**Vẽ sơ đồ Container:** Web App, Mobile App, Database, Payment Service. 19](#_Toc209608802)

[**Vẽ sơ đồ Component:** Bên trong Web App, có module tìm kiếm, module mượn/trả, module quản lý tài khoản. 20](#_Toc209608803)

[**Vẽ sơ đồ Code:** Ví dụ class diagram cho module “Quản lý mượn sách”. 21](#_Toc209608804)

[ ***Các quyết định thiết kế chính của sơ đồ:*** 23](#_Toc209608805)

1. **BÀI TẬP MINH HỌA**
2. **Vẽ lại sơ đồ kiến trúc SPA (Single Page Application).**

A screenshot of a computer

Description automatically generated

*Giải Thích :*

Đây là sơ đồ mô tả **kiến trúc tổng thể của một hệ thống thương mại điện tử**

(E-commerce) với quy trình từ phía **client**người dùng) đến **cơ sở dữ liệu (RDBMS)**:

* Phía Fontend : SPA (single page application) giảm tải backend. Xử lý logic phía client. Load Balencer mục đích là phân phối request của người dùng đến server.
* Phía Backend : được chia thành nhiều module
  + Product : (listing products) phải đảm bảo UI thân thiện, Logic nghiệp vụ đúng chuẩn và truy vấn dữ liệu từ sản phẩm.
  + Basket : (add to basket) UI thể hiện giỏ hàng. Logic nghiệp vụ cơ bản thêm/sửa/xoá sản phẩm trong giỏ hàng. Ghi/đọc dữ liệu giỏ hàng.
  + Order : (checkout order ) Clean Architecture phân tách các lớp rõ ràng (lớp View, Lớp Model, Lớp Controller)

1. **Vẽ sơ đồ triển khai CI/CD.**

A screenshot of a computer

Description automatically generated

*Giải Thích :*

Sơ đồ trên dựa vào DevOps Cycle :

* + Development 🡪 Build 🡪 Deploy
* Trong quy trình Development : Dev (lập trình viên) chọn ngôn ngữ C# để phát triển thông qua Visual Studio sau đó push code lên GITHUB đồng thời chuẩn bị Docker File
* Trong quy trình Build : Travis CI chạy pineline : lấy code từ github, build image docker bằng Docker File và đẩy lên Docker Hub.
* Trong quy trình Deploy/Shift : Kubernetes triển khải, quản lý scale, load balancing, và giám sát trạng thái container.

1. **Vẽ sơ đồ API của hệ thống.**

A white background with black lines

AI-generated content may be incorrect.

A group of colored lines

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A close-up of a line

AI-generated content may be incorrect.

1. **Vẽ sơ đồ C1 - System Context và giải thích.**

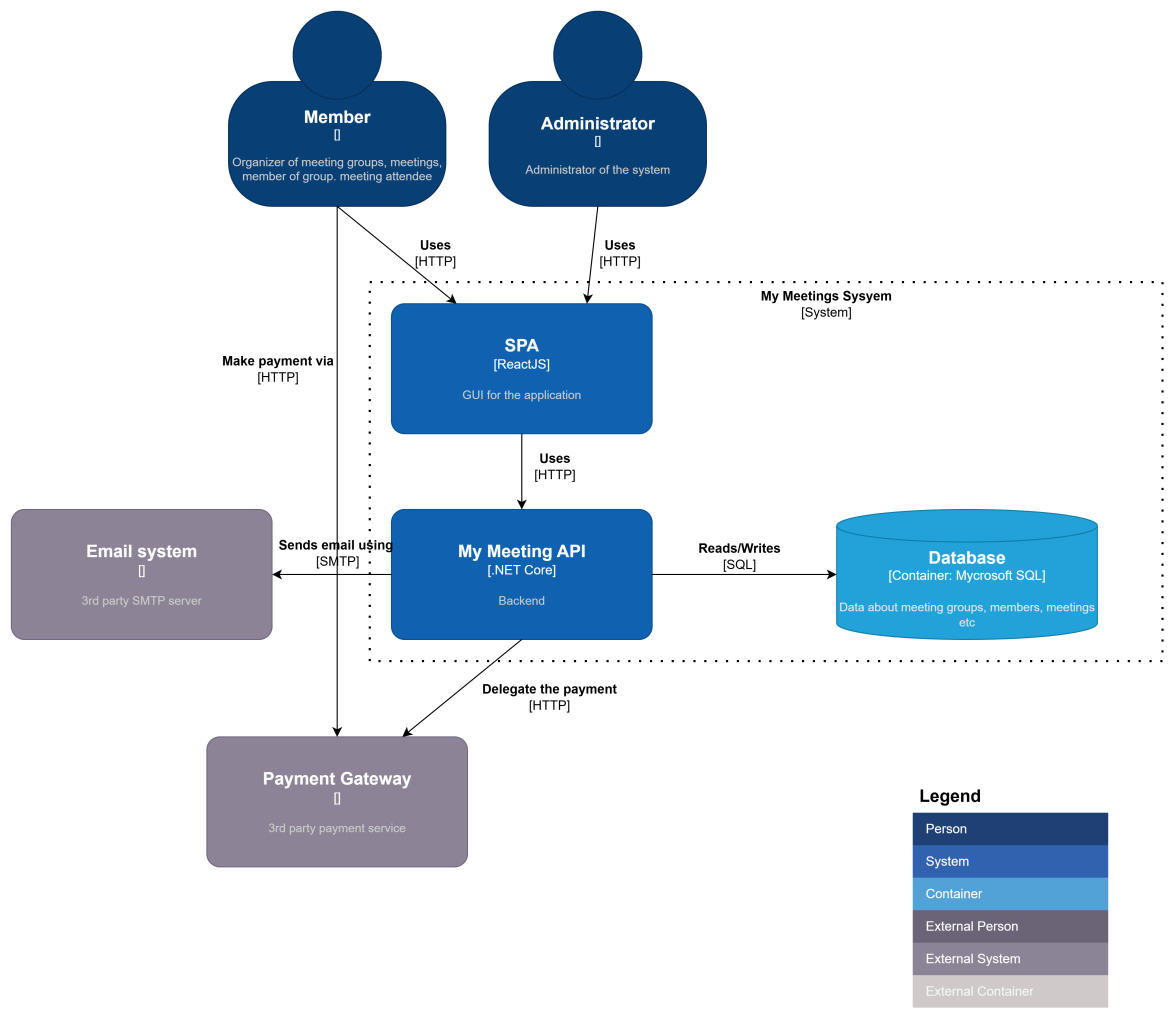
A screenshot of a computer

Description automatically generated

*Giải Thích*

* Hệ Thống C1 – System Context tuân theo nguyên tắc
  + Ai là người sử dụng hệ thống ?
  + Hệ thống nào tương tắc với nó ?
  + Luồng hệ thống dữ liệu và luồng qua lại
  + Không đi sâu vào chi tiết kỹ thuật, database hay API nội bộ.
* Quy trình :
  + **My meetings (System):** Hệ thống trung tâm, nơi thực hiện quản lý họp, nhóm họp, thanh toán và gửi email.
  + **Member (External person):** Người dùng bên ngoài, có thể tổ chức, tham giahọp,và thanh toán.
  + **Administrator (External person):** Quản trị viên, quản lý thành viên, nhóm họpvà cuộc họp.
  + **Payment Gateway (External system):** Xử lý thanh toán cho hệ thống.
  + **Email Service (External system):** Gửi email thông báo cho người dùng.

1. **Vẽ sơ đồ C2 – Container và giải thích:**



*Giải thích:*

1. Các tác nhân trong hệ thống:

* **Member**: Người dùng hệ thống, có thể tổ chức nhóm họp, tham gia nhóm, tham gia cuộc họp.
* **Administrator**: Quản trị viên hệ thống, chịu trách nhiệm cấu hình, quản lý hệ thống.

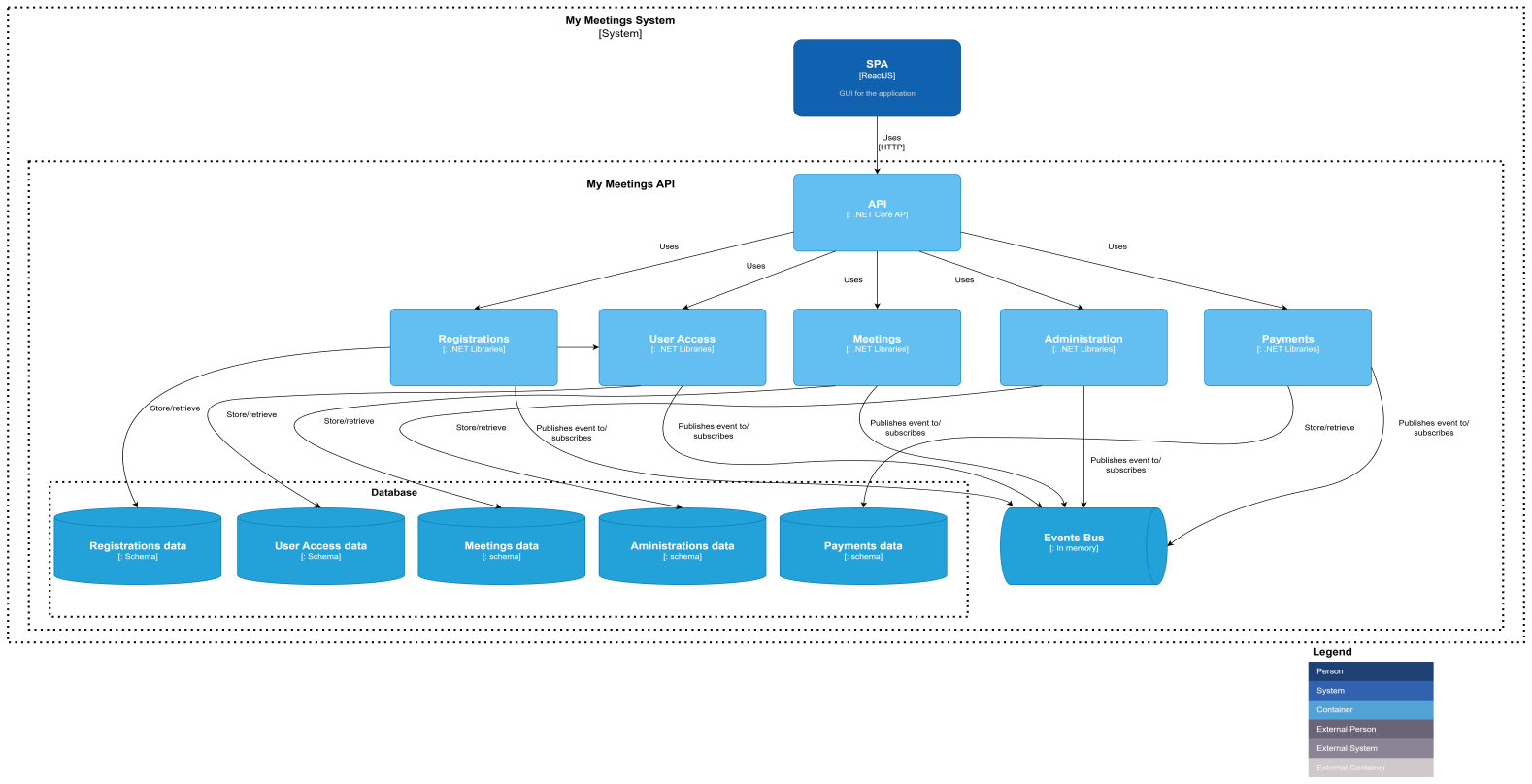
1. Các Container trong hệ thống:

* SPA (ReactJS):
  + Giao diện người dùng của ứng dụng (UI).
  + Được xây dựng dưới dạng Single Page Application để tăng tốc độ phản hồi, giảm tải cho server.
  + Cung cấp giao diện cho cả Member và Administrator.
* My Meetings API (.NET Core):
  + Backend chính của hệ thống.
  + Nhận yêu cầu từ SPA qua HTTP.
  + Xử lý nghiệp vụ: quản lý nhóm, lịch họp, người tham gia, gửi thông báo, tích hợp thanh toán.
* Database (Microsoft SQL):
  + Lưu trữ toàn bộ dữ liệu: nhóm họp, thành viên, cuộc họp, lịch sử tham gia.
  + API thực hiện thao tác đọc/ghi dữ liệu qua SQL.

1. Các tác nhân bên ngoài:
   * Email System (SMTP server của bên thứ 3): Dùng để gửi email thông báo đến người dùng (nhắc lịch họp, mời tham gia, xác nhận đăng ký...).
   * Payment Gateway (3rd party payment service):
   * Cổng thanh toán bên ngoài.
   * Khi người dùng cần trả phí (ví dụ: phí dịch vụ), API sẽ chuyển tiếp yêu cầu thanh toán đến Payment Gateway.
2. Luồng tương tác chính

* Member/Administrator → truy cập SPA (ReactJS) bằng trình duyệt.
* SPA → gửi request đến My Meetings API để lấy dữ liệu hoặc thực hiện nghiệp vụ.
* My Meetings API ↔ Database: lưu trữ & truy xuất dữ liệu.
* My Meetings API → Email System: gửi email qua giao thức SMTP.
* My Meetings API → Payment Gateway: thực hiện giao dịch thanh toán.
* Sơ đồ trên minh họa kiến trúc Container của hệ thống My Meetings, trong đó người dùng và quản trị viên tương tác qua SPA; logic nghiệp vụ xử lý ở API backend; dữ liệu nằm trong Database; và các dịch vụ bên ngoài như Email System & Payment Gateway được tích hợp để hỗ trợ đầy đủ chức năng.

1. **Vẽ sơ đồ C3 – Component (High-Level) và giải thích:**



*Giải thích:*

**Đây là Container Diagram**, chi tiết hơn ở mức chia nhỏ API và Database

* SPA (ReactJS): Đây là ứng dụng web dạng Single Page Application, đóng vai trò GUI cho toàn hệ thống. Người dùng sẽ truy cập vào đây để thao tác.
* API (.NET Core API):
  + Là lớp trung gian, đóng vai trò cầu nối giữa SPA và các dịch vụ bên dưới.
  + SPA sẽ không kết nối trực tiếp database mà chỉ gọi đến API này.
* Hệ thống có nhiều dịch vụ (Services / Containers)riêng biệt, triển khai độc lập (theo hướng microservices):
  + **Registrations** (.NET Core microservice): Xử lý nghiệp vụ đăng ký.
  + **User Access** (.NET Core microservice): Quản lý tài khoản, phân quyền người dùng.
  + **Meetings** (.NET Core microservice): Quản lý cuộc họp.
  + **Administration** (.NET Core microservice): Chức năng quản trị, cấu hình hệ thống.
  + **Payments** (.NET Core microservice): Quản lý thanh toán.
* Mỗi dịch vụ có database riêng (theo mô hình Database per Service):
  + **Registrations data**
  + **User Access data**
  + **Meetings data**
  + **Administration data**
  + **Payments data**
* Ngoài ra còn có:
  + Events Bus (External System): Hệ thống bus sự kiện, giúp các microservice giao tiếp bất đồng bộ với nhau (event-driven architecture).
* Kết nối giữa các thành phần:
  + Người dùng → SPA (ReactJS) → API → Các microservice.
  + Mỗi microservice truy cập database riêng của mình (không chia sẻ DB).
  + Các microservice có thể trao đổi thông tin qua Event Bus.

1. **Vẽ sơ đồ C3 – Component (Module-Level) sau:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

1. **Vẽ sơ đồ xử lý 1 request:**

A diagram of a software system

AI-generated content may be incorrect.

1. **BÀI TẬP ỨNG DỤNG**
2. **Mô tả:** Xây dựng hệ thống cho phép người dùng mượn/trả sách, quản lý kho sách, và thanh toán phí trễ hạn.

**Tổng quan kiến trúc — Hệ thống Quản lý Mượn/Trả Sách (Modular Monolith với DDD)**

**Mục tiêu:** Xây dựng hệ thống cho phép người dùng mượn/trả sách, quản lý kho sách, và thanh toán phí trễ hạn — thiết kế theo tinh thần Modular Monolith + DDD (tham khảo cấu trúc C1–C4 của Kamil Grzybek).

**C1 — System Context (tóm tắt)**

Hệ thống là một **Modular Monolith** phục vụ miền thư viện: người dùng (patron) đăng ký tài khoản, tra cứu catalog, đặt/mượn/trả sách, quản lý kho (bản sao sách — copies), xử lý phạt do trả muộn, và các tác vụ quản trị (thêm sách, nhập kho, điều chỉnh phí).

**Actor chính:**

* **Patron (User):** tìm sách, đặt trước, mượn, trả, xem lịch sử, thanh toán phạt.
* **Librarian / Admin:** quản lý catalog, nhập/xuất kho, duyệt đặt trước, cấu hình chính sách (số ngày mượn, phí trễ).
* **Payer (Payment gateway):** xử lý thanh toán phạt muộn/chi phí dịch vụ.
* **External Systems:** hệ thống email/SMS cho thông báo, hệ thống báo cáo/analytics.

**Module cao cấp (tính năng):** User Access, Catalog, Borrowing (Loans), Inventory, Reservations, Payments, Notifications, Administration. API mỏng ở phía trước; tích hợp module-to-module qua In-Memory Event Bus (Pub/Sub).

**C2 — Container View**

**API (REST / GraphQL)**

* **Vai trò:** nhận HTTP request → xác thực/ủy quyền (dựa vào User Access) → gửi Command/Query tới module đích → trả response. Không chứa business logic. API giao tiếp với module qua các ports (Command/Query interfaces).

**Các Module nghiệp vụ (chạy trong cùng tiến trình — modular monolith):**

* **User Access:** đăng ký, xác thực, quản lý vai trò/permission.
* **Catalog:** metadata sách (title, authors, ISBN, subjects), tìm kiếm, đề xuất, tags.
* **Inventory:** quản lý bản sao sách (BookCopy), trạng thái (Available/OnLoan/Reserved/Lost/Damaged), location (shelf), nhập/ghi mã vạch.
* **Borrowing (Loans):** quản lý luồng mượn-trả, tính hạn trả (due date), phát sinh phí trễ khi trả muộn, xử lý gia hạn (renewal) và trả sớm.
* **Reservations:** đặt trước khi tất cả bản sao đang cho mượn; quản lý hàng đặt, cảnh báo hết hạn hold.
* **Payments:** tạo invoice/phạt, tích hợp cổng thanh toán, ghi lịch sử giao dịch.
* **Notifications:** gửi email/SMS/push thông báo nhắc hạn trả, thông báo sẵn sàng khi sách đến lượt.
* **Administration:** bảng điều khiển quản trị, cấu hình chính sách mượn/phạt, báo cáo.
* **In-Memory Events Bus:** Pub/Sub nội bộ để module publish Integration Events và các module khác subscribe.

**Nguyên tắc tích hợp & giả định:**

* API không chứa business logic;
* API ↔ Module qua interface nhỏ (Commands/Queries);
* Module-to-Module **chỉ** qua Integration Events; không đọc chéo DB;
* Mỗi module sở hữu schema riêng (có thể tách DB khi cần);
* Chỉ 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 surface nhỏ nhất.

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

Mỗi module tuân theo Clean Architecture & DDD, chia thành các phần sau:

* **Application:** handlers cho Command/Query, orchestration use-cases, validation, mapping to/from DTO, phát domain events → publish Integration Events.
* **Domain:** Entities, Value Objects, Aggregates, Domain Services, Domain Events — *persistence-ignorant*, giàu hành vi.
* **Infrastructure:** ORM (EF Core), Repositories (implementation), access to Event Bus, scheduling (Quartz jobs) cho việc kiểm tra due date và áp phạt tự động, integration with external payment/email systems.
* **IntegrationEvents:** contracts cho events public (ví dụ BookReturned, LoanOverdueDetected, ReservationFulfilled) — đây là điểm duy nhất module khác phụ thuộc.

**Pattern vận hành:** CQRS ở mức use-case (Command handlers thay đổi state; Query handlers đọc từ read-model/DB). Domain phát Domain Events nội bộ; Application chuyển sang Integration Events để publish lên Event Bus.

**Ví dụ components quan trọng (Borrowing module):**

* Aggregates: Loan (aggregate root) — chịu trách nhiệm tạo loan, gia hạn, nhận trả, tính phí trễ;
* Repositories: ILoanRepository (Domain interface), EfLoanRepository (Infrastructure impl);
* Application services: CreateLoanCommandHandler, ReturnLoanCommandHandler, RenewLoanCommandHandler;
* Integration Events emitted: LoanCreated, LoanReturned, LoanOverdue.

**C4 — Code-level (ví dụ aggregate & luồng)**

**Ví dụ Aggregate: Loan**

* **State:** loanId, patronId, bookCopyId, loanDate, dueDate, returnedDate, status, accumulatedLateFees
* **Behavior (methods):** StartLoan(), Return(returnDate), Renew(), MarkOverdue(checkDate)
* **Quy tắc nghiệp vụ:**
  + Khi Return() và returnDate > dueDate ⇒ tính lateDays = (returnDate - dueDate), tính fee = lateDays \* perDayFee, domain emit LoanReturned chứa lateFee;
  + Renew() chỉ cho phép khi sách chưa bị reservation khác và số lần renew trong policy;
  + MarkOverdue() được job background (Quartz) hoặc scheduler gọi để publish LoanOverdue Integration Event.

**Luồng xử lý mượn-trả (Request & CQRS)**

1. API nhận request mượn sách → xác thực bởi User Access → gửi CreateLoanCommand tới Borrowing.Application.
2. CreateLoanCommandHandler kiểm tra availability via Inventory.Repository (qua interface/integration event nếu cần) → domain Loan.StartLoan() → lưu vào LoanRepository → emit Domain Event LoanStarted.
3. Application chuyển Domain Event thành Integration Event LoanCreated và publish lên In-Memory Event Bus.
4. Inventory module subscribe LoanCreated để cập nhật trạng thái BookCopy -> OnLoan.
5. Khi đến hạn trả, Quartz job kiểm tra loans sắp quá hạn hoặc đã overdue → gọi domain method MarkOverdue() → publish LoanOverdue → Payments/Notifications phản ứng (tạo invoice, gửi nhắc nhở).

**Dữ liệu & biên giới module**

* **Data ownership:** mỗi module quản lý schema riêng (ví dụ Borrowing schema, Inventory schema, Catalog schema). Không truy vấn chéo DB giữa module.
* **Synchronization:** state sharing via Integration Events (ví dụ BookCopyStatusChanged, LoanCreated).

**Bảo mật & khởi tạo**

* **AuthN/AuthZ:** User Access module chịu trách nhiệm; API kiểm tra token/session trước mọi Command/Query.
* **Khởi tạo module:** API gọi Initialize(...) cho từng module trong Startup (hoặc DI bootstrapper), truyền cấu hình (connection strings, email settings, payment credentials), dựng Composition Root riêng cho module, khởi động Quartz jobs và đăng ký event handlers với In-Memory Event Bus.

**Kiểm thử & chất lượng**

* **Unit tests:** domain rules (Loan renew rules, late fee calculation, reservation precedence).
* **Integration tests:** module-level interactions (Loan ↔ Inventory via Integration Events), persistence tests (migrations), payment gateway stubs.
* **System tests / E2E:** simulating user flows (mượn → trả trễ → generate invoice → thanh toán).
* **CI/CD:** schema migrations, static analysis, mutation testing, contract tests cho IntegrationEvents.

**Các quyết định thiết kế & giả định chính**

* Là monolith để đơn giản hoá deployment nhưng với đóng gói chặt để có thể tách module ra microservice khi cần.
* Event-Driven Architecture nội bộ (In-Memory Event Bus) để giảm coupling; không cho phép gọi method chéo module để giữ tính độc lập dữ liệu.
* CQRS ở mức use-case cho phép tối ưu hoá read-model (materialized views) phục vụ search và danh mục.
* Chính sách phạt có thể cấu hình (per-day fee, grace period) và được lưu ở module Administration.

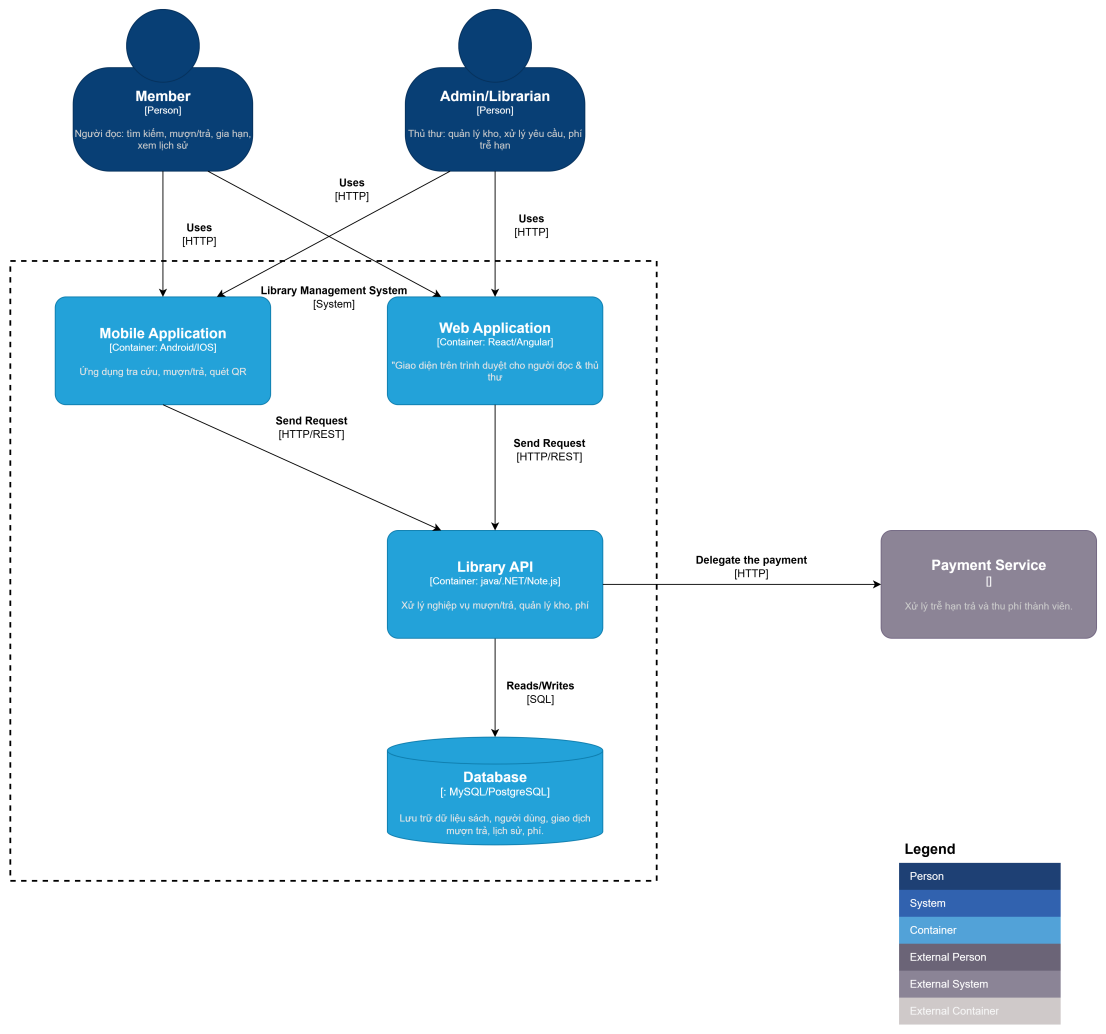
1. **Yêu cầu:**

### **Vẽ sơ đồ Context:** Thư viện online, Người dùng, Thủ thư, Hệ thống thanh toán.

**A group of blue and grey rectangular shapes

Description automatically generated**

### **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.

A diagram of a company

AI-generated content may be incorrect.

### **Vẽ sơ đồ Code:** Ví dụ class diagram cho module “Quản lý mượn sách”.

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a diagram

AI-generated content may be incorrect.**

* ***Các quyết định thiết kế chính của sơ đồ:***

1. Tách Mobile Application và Web Application để:

* Người dùng có nhiều kênh truy cập khác nhau: đọc giả có thể tra cứu/mượn trả bằng điện thoại, thủ thư thì thường làm việc trên máy tính.
* Trải nghiệm người dùng được tối ưu riêng: Mobile App tiện lợi cho quét QR, thông báo nhanh; Web App phù hợp cho các tác vụ phức tạp như quản lí kho, xử lý yêu cầu.
* Ưu điểm:
  + Giao diện linh hoạt, phù hợp nhu cầu của từng người dùng.
  + Có thể triển khai, bảo trì và cập nhật độc lập.

1. Dùng SPA (Web Application với React/Angular)

* SPA (Single Page Application) cho trải nghiệm mượt mà hơn, hạn chế tải lại toàn bộ trang.
* Tăng tốc độ thao tác cho thủ thư và người đọc khi tra cứu/mượn trả nhiều lần.
* Ưu điểm:
  + Cải thiện UX.
  + Dễ dàng tái sử dụng API backend.

1. Sử dụng Library API làm trung tâm (Java/.NET/Node.js)

* Backend tách biệt với giao diện, cho phép nhiều client (web, mobile, hệ thống ngoài) cùng truy cập.
* Quản lý nghiệp vụ mượn/trả, phí, quản lý kho… tập trung, thống nhất logic.
* Ưu điểm:
  + Giảm lặp lại code nghiệp vụ.
  + Dễ mở rộng khi thêm ứng dụng mới (ví dụ kiosk tự mượn sách).

1. Một Database chung (MySQL/PostgreSQL)

* Hệ thống quản lý thư viện chưa quá lớn nên một database quan hệ trung tâm là hợp lý, đảm bảo tính toàn vẹn dữ liệu (foreign key, transaction…).
* Dễ dàng cho báo cáo, thống kê.
* Ưu điểm:
  + Quản lý dữ liệu tập trung.
  + Hỗ trợ tốt cho giao dịch (mượn sách, trả sách, thu phí).