

Báo cáo kết thúc môn CÔNG NGHỆ PHẦN MỀM

Đề tài:

HỆ THỐNG ỨNG DỤNG ĐẶT BÀN NHÀ HÀNG

Giảng viên hướng dẫn:

Nguyễn Bảo Ân

Sinh viên thực hiện:

Nguyễn Trọng Đạt	110122217
Lê Khánh Duy	110122002
Nguyễn Huỳnh Chí	110122001

LÝ DO CHỌN ĐỀ TÀI

&

MỤC TIÊU

Trong bối cảnh chuyển đổi số mạnh mẽ, trải nghiệm ẩm thực không chỉ dừng ở chất lượng món ăn mà còn đòi hỏi sự tiện lợi trong đặt bàn và phục vụ. Hình thức đặt bàn truyền thống như gọi điện hay đến trực tiếp còn nhiều bất cập, nhất là vào giờ cao điểm. Trong khi đó, người dùng ngày càng ưa chuộng các nền tảng số có khả năng đặt bàn linh hoạt, tra cứu nhanh và thanh toán tiện lợi. Vì vậy, nhóm chọn đề tài “**Xây dựng hệ thống đặt bàn nhà hàng**” nhằm tối ưu trải nghiệm cho cả khách hàng và nhà hàng.

Đối với người dùng: Giao diện thân thiện, bắt mắt thu hút người dùng xem thông tin về nhà hàng, đặt bàn dễ dàng, đặt theo mô tả bàn mong muốn và theo dõi lịch sử đặt bàn, tiết kiệm thời gian và tránh phải chờ đợi.

Đối với nhà hàng: Hệ thống cập nhật tình trạng bàn theo thời gian thực, hỗ trợ phân bổ không gian và nhân sự hiệu quả, giảm tải cho lễ tân và nâng cao chất lượng phục vụ.

Đối với người quản lý: Ứng dụng giúp lọc và hiển thị bàn theo khung giờ, hỗ trợ việc tổ chức, điều phối bàn trống hợp lý và đảm bảo vận hành trơn tru.

PHÂN TÍCH YÊU CẦU HỆ THỐNG

1

CHỨC NĂNG TỔNG
QUAN CỦA HỆ THỐNG

2

CÁC PHI CHỨC NĂNG
CỦA HỆ THỐNG

3

MÔ HÌNH PHÁT TRIỂN
PHẦN MỀM

1. CHỨC NĂNG TỔNG QUAN CỦA HỆ THỐNG

Chức năng dành cho khách hàng

Đăng ký và tạo tài khoản:

- Người dùng đăng ký với thông tin như: tên người dùng, họ tên, mật khẩu, email, số điện thoại, ngày sinh và giới tính. Đăng nhập với tên người dùng và mật khẩu.
- Sau khi đăng nhập có thể sử dụng được các chức năng chính.

Đặt bàn, xem bàn đã đặt, hủy bàn, xem lịch sử đặt bàn:

- Đặt bàn đơn giản bằng cách chọn chi nhánh nhà hàng, số lượng người, thời gian nhận bàn và mô tả vị trí bàn.
- Xem lại thông tin bàn đã đặt, theo dõi trạng thái bàn để biết được đơn đặt bàn đã được xác nhận bàn chưa.
- Hủy bàn đã khi không còn nhu cầu với điều kiện trước một tiếng.
- Xem lịch sử toàn bộ các đơn đặt bàn.

Chức năng dành cho quản lý

Cũng bao gồm các chức năng dành cho khách hàng, nhưng với tài khoản dành cho quản lý thì sẽ có thêm các chức năng:

- Duyệt bàn: chọn bàn phù hợp và xác nhận bàn cho các đơn đặt.
- Lọc các bàn đã được duyệt, hỗ trợ cho việc tìm bàn trống hay tính toán thời gian phù hợp để chọn bàn cho các đơn đặt chưa được duyệt.

2. CÁC PHI CHỨC NĂNG CỦA HỆ THỐNG

Ngoài các chức năng chính được tương tác qua giao diện của hệ thống thì cũng có những phi chức năng đáng đề cập đến:

- ▶ **Hiệu suất:** Hệ thống phản hồi nhanh khi cập nhật, đặt bàn hoặc tra cứu thông tin. Trang tải mượt, không xảy ra treo hoặc giật lag.
- ▶ **Tiện lợi:** Người dùng đã đăng nhập trên trình duyệt sẽ được giữ trạng thái đăng nhập, sử dụng ngay mà không cần đăng nhập lại.
- ▶ **Khả năng mở rộng:** Thiết kế tách biệt frontend (ReactJS) và backend (Spring Boot), dễ mở rộng và tích hợp app. Dùng Docker giúp triển khai linh hoạt.
- ▶ **Bảo mật:** Mật khẩu được mã hóa bằng Bcrypt, xác thực API bằng JWT. Chức năng admin chỉ cho phép tài khoản có vai trò quản lý truy cập.
- ▶ **Sẵn sàng:** Hệ thống có thể hoạt động 24/7, xử lý nhiều lượt truy cập cùng lúc mà không bị chậm trễ.
- ▶ **Duy trì & phát triển:** Mã nguồn tổ chức theo RESTful API, dễ bảo trì và phát triển. Frontend và backend tách biệt để làm việc nhóm hiệu quả, kiểm thử liên tục bằng Postman.

3. MÔ HÌNH PHÁT TRIỂN CỦA HỆ THỐNG

Để phát triển hệ thống nhóm sử dụng mô hình Agile cụ thể với quy trình Scrum là phương pháp phát triển linh hoạt, chia quá trình phát triển thành các vòng lặp ngắn gọi là sprint. Mô hình này cho phép nhóm dự án liên tục nhận phản hồi và điều chỉnh hệ thống theo thực tế, phù hợp với các dự án có yêu cầu thay đổi thường xuyên và cần cải tiến liên tục.

THIẾT KẾ HỆ THỐNG

1. Kiến trúc tổng thể

Client (Front-end)

Sử dụng ReactJS + Tailwind CSS để xây dựng giao diện người dùng để sử dụng trực quan, tương tác với các chức năng hệ thống không bị phức tạp.

Gửi yêu cầu đến server bằng cách sử dụng phương thức HTTP (GET, POST, PUT) thông qua API RESTful.

Server (Back-end)

Kiến trúc phân lớp trong Spring Boot giúp tổ chức mã nguồn rõ ràng, tách biệt trách nhiệm giữa các phần, hỗ trợ bảo trì, mở rộng và kiểm thử dễ dàng.

Controller layer tiếp nhận HTTP request, gọi service xử lý và trả về response phù hợp cho client.

Service layer xử lý logic nghiệp vụ, kiểm tra dữ liệu, gọi repository để thao tác CRUD với cơ sở dữ liệu.

Repository layer quản lý truy cập dữ liệu, dùng JPA hoặc ORM để thao tác với cơ sở dữ liệu thông qua các đối tượng Java.

Entity layer đại diện bảng dữ liệu dưới dạng đối tượng, giúp ánh xạ giữa thuộc tính và cột, hỗ trợ ràng buộc và quan hệ giữa các thực thể.

THIẾT KẾ HỆ THỐNG

1. Kiến trúc tổng thể

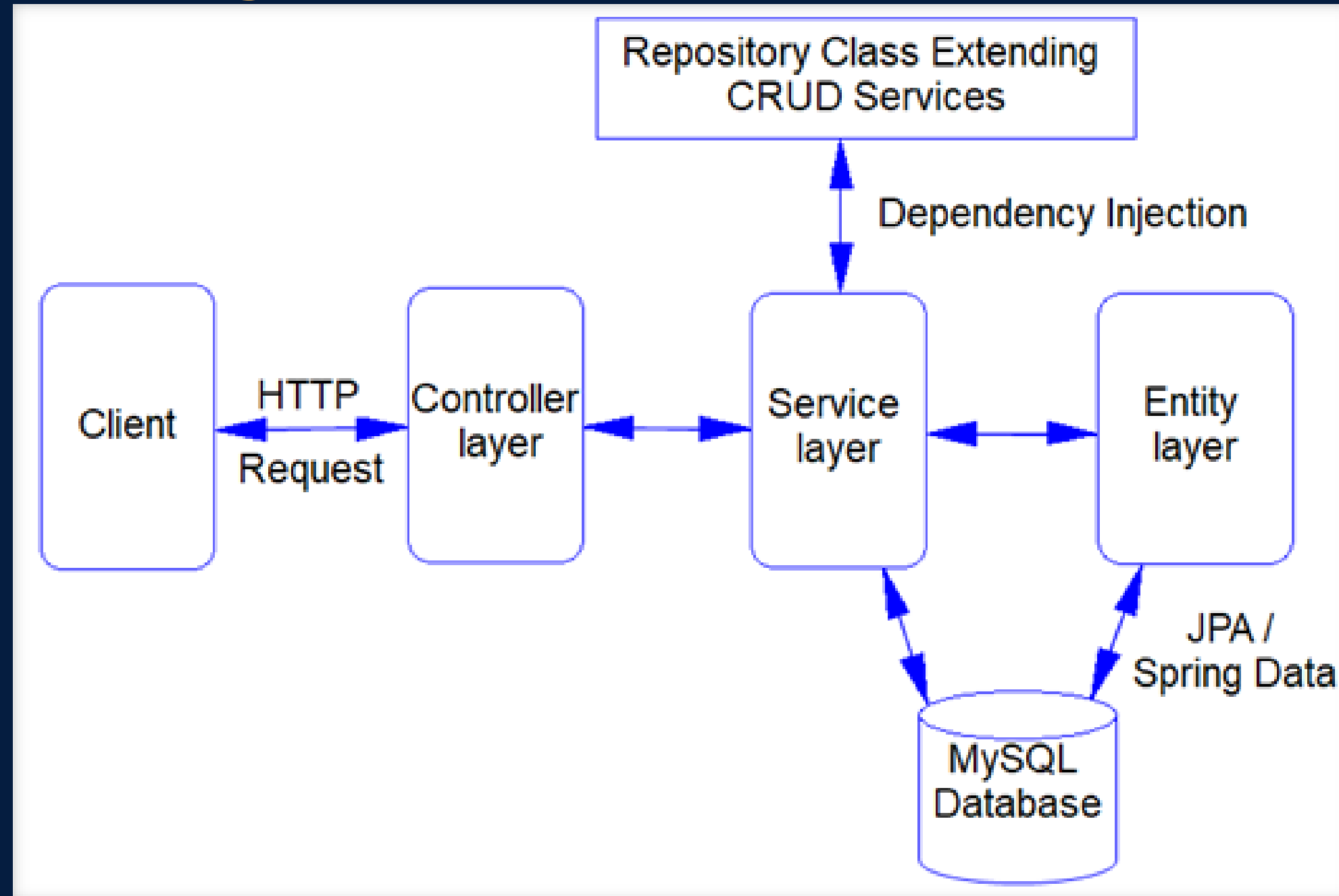
Cơ sở dữ liệu

Hệ thống sử dụng image MySQL 8.0 - phiên bản chính thức từ Docker Hub làm hệ quản trị cơ sở dữ liệu chính. Backend sử dụng JPA / Spring Data để ánh xạ các entity Java với các bảng và dễ dàng tương tác với cơ sở dữ liệu.

Với mỗi truy vấn hay cập nhật cơ sở dữ liệu đều được thực hiện qua các phương thức như `save()`, `findById()`, `findAll()`, `deleteById()`, ... mà không cần viết câu lệnh SQL.

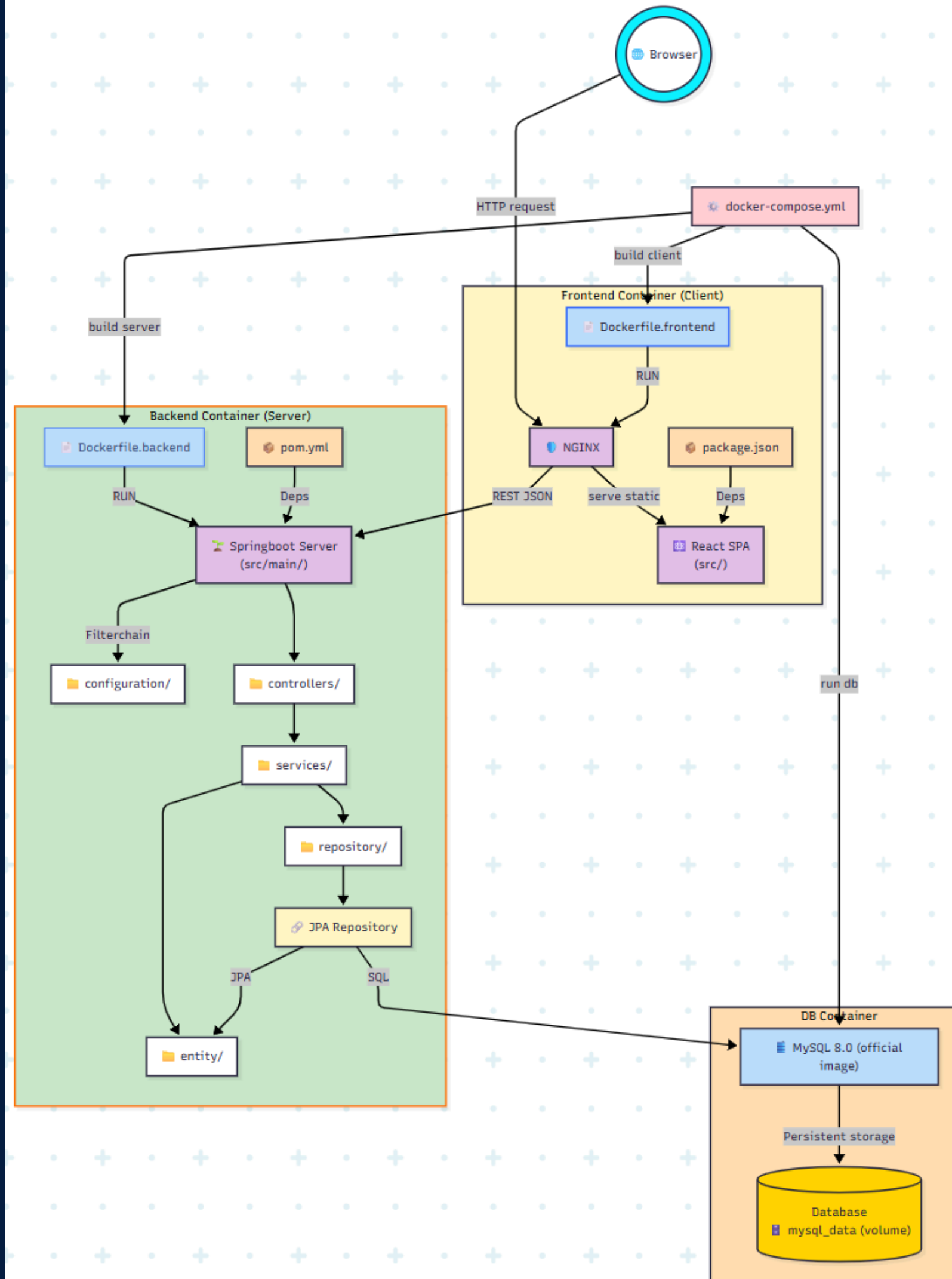
THIẾT KẾ HỆ THỐNG

1. Kiến trúc tổng thể



Sơ đồ minh họa tổng quát kiến trúc phân lớp

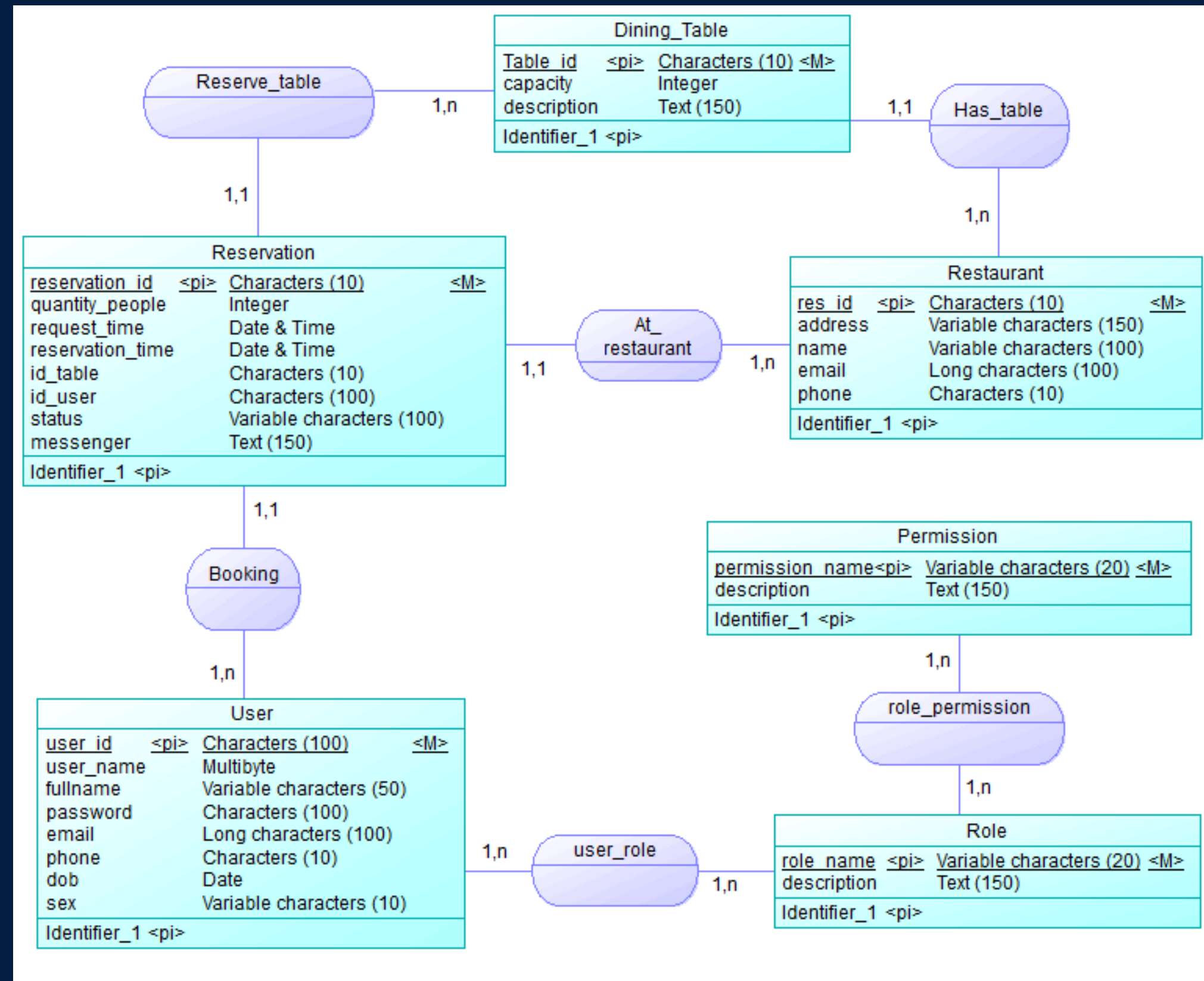
Sơ đồ kiến trúc tổng thể



THIẾT KẾ HỆ THỐNG

2. Thiết kế cơ sở dữ liệu

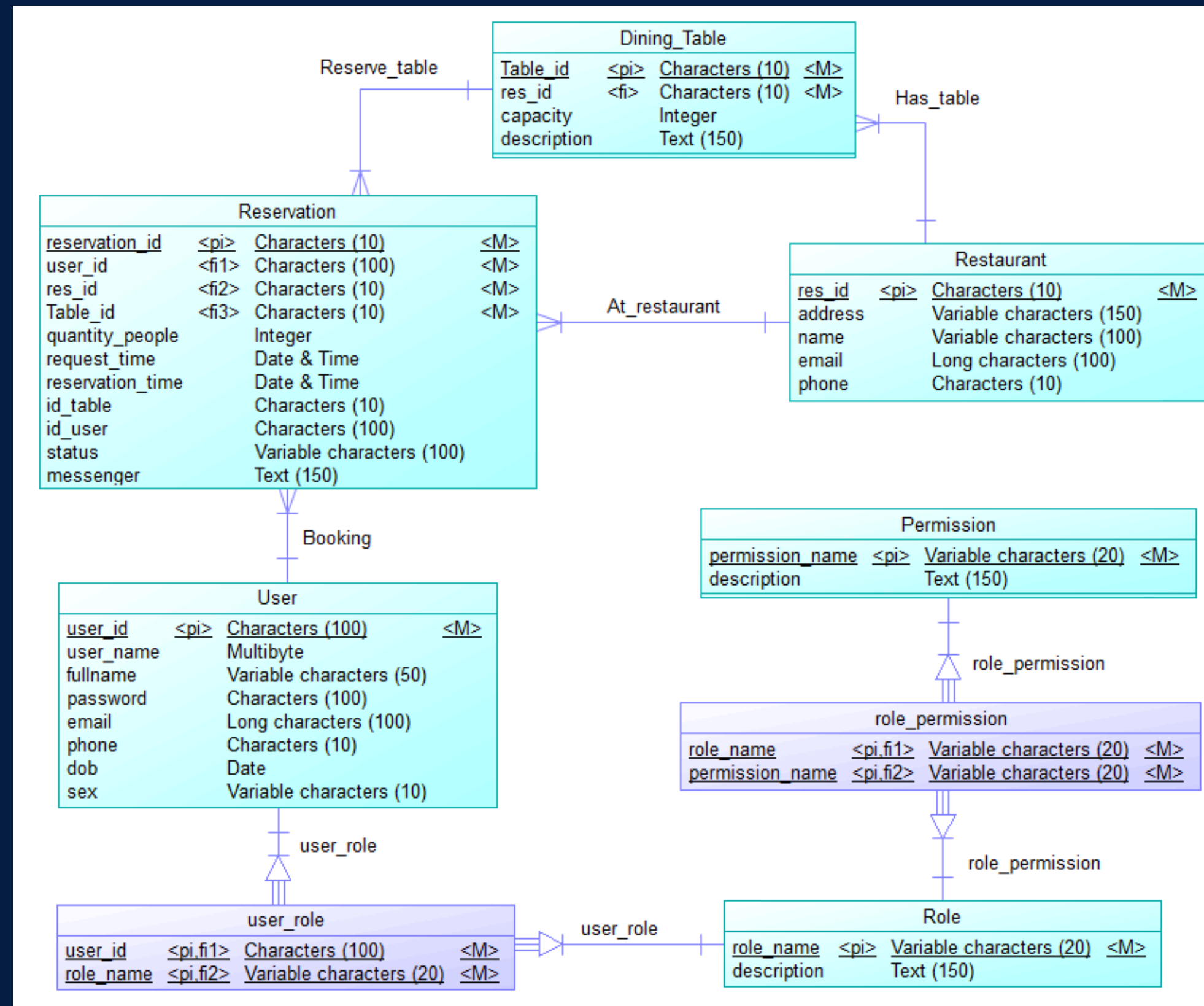
Mô hình quan niệm dữ liệu (ERD)



THIẾT KẾ HỆ THỐNG

2. Thiết kế cơ sở dữ liệu

Mô hình vật lý



THIẾT KẾ HỆ THỐNG

3. Thiết kế API

Hệ thống áp dụng kiến trúc RESTful API theo chuẩn OpenAPI 3.0, được tài liệu hóa rõ ràng bằng Swagger để hỗ trợ phát triển và tích hợp, đồng thời sử dụng Postman để kiểm thử và xác thực tính đúng đắn của các API trong quá trình phát triển.

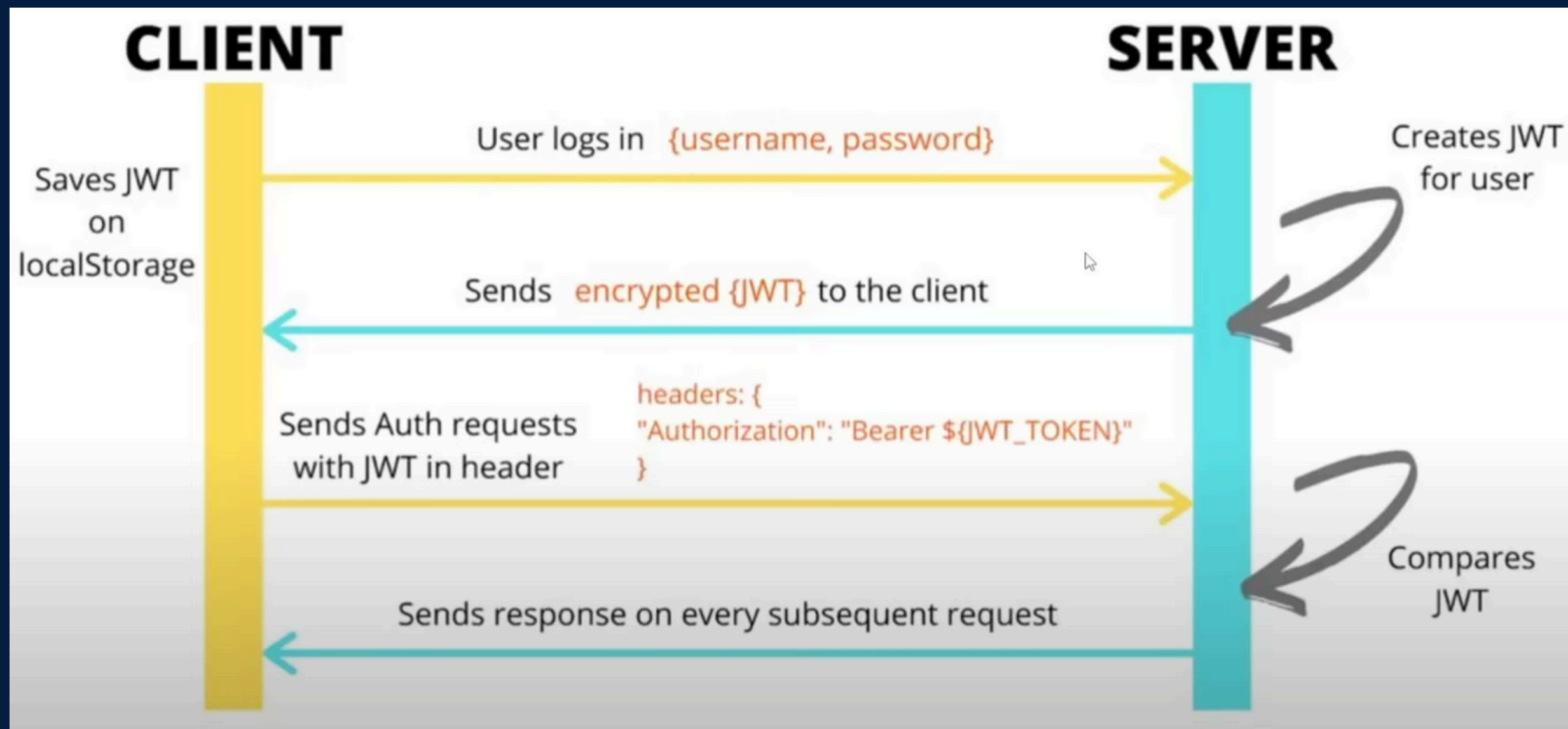
Thông tin cấu hình API:

- Context-path: /restaurant
- Base URL: `http://localhost:8386/restaurant/`
- Xác thực: Bearer Token (JWT) – header Authorization: Bearer <token>
- Định dạng dữ liệu (request và response): JSON

THIẾT KẾ HỆ THỐNG

3. Thiết kế API

Hầu hết các chức năng gọi API để client tương tác với server đều được xác thực thông qua JWT Token. Sau khi người dùng đăng nhập sẽ trả về token và được lưu trữ trong localStorage của trình duyệt, khi gọi API token sẽ kèm trong các request thông qua header Authorization.



Quy trình xác thực người dùng bằng JWT

THIẾT KẾ HỆ THỐNG

3. Thiết kế API

Nhóm chức năng API	Path prefix	[method] /endpoint
User	/users	[GET] /, [POST] /, [GET] /getByID/{userID}, [PUT] /{userID}, [GET] /myInfo
Authentication	/auth	[POST] /token, [POST] /introspect, [POST] /refresh, [POST] /logout
Restaurant	/restaurant	[POST] /, [GET] /, [DELETE] /delete/{idrestaurant}
Table	/table	[POST] /, [GET] /, [DELETE] /delete/{idTable}
Reservation	/reservation	[POST] /, [GET] /, [GET] /history, [GET] /coming, [GET] /unconfirmed, [POST] /filter, [PUT] /confirm/reservationId, [PUT] /cancle/reservationId

THIẾT KẾ HỆ THỐNG

3. Thiết kế API

Ví dụ về API chức năng đăng nhập người dùng

- Phương thức: POST
- Endpoint: `http://localhost:8386/restaurant/users`

Request body:

```
{
  "username": "john_doe",
  "fullname": "John Doe",
  "password": "Password123",
  "email": "johndoe@example.com",
  "phone": "0123456789",
  "dob": "2000-01-01",
  "sex": "male"
}
```

Response:

Code	Details
200	<div>Response body</div> <pre>{ "code": 1000, "result": { "id": "4680197c-2224-453c-81b4-d0349cb9ac3b", "username": "john_doe", "fullname": "John Doe", "email": "johndoe@example.com", "phone": "0123456789", "dob": "2000-01-01", "sex": "male", "roles": [{ "name": "USER", "description": "Vai trò User có quyền: tạo TK, sửa TT, xem TT, thêm DLĐB, xem TT&DL, hủy ĐB", "permissions": [{ "name": "CREATE-RESERVATION", "description": "Tạo một dữ liệu đặt bàn" }, { "name": "CREATE-DATA", "description": "Tạo (thêm) dữ liệu" }, { "name": "GET-RESERVATED-HISTORY", "description": "Xem lịch sử đặt bàn" }] }] } }</pre>

TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

1. CÔNG NGHỆ SỬ DỤNG
2. QUY TRÌNH CI/CD VỚI GITHUB ACTION
3. CẤU HÌNH DOCKER VÀ TRIỂN KHAI

1. CÔNG NGHỆ SỬ DỤNG

Công nghệ sử dụng phát triển Front-end

Sử dụng ReactJS để phát triển giao diện người dùng theo mô hình Single Page Application (SPA), giúp ứng dụng phản hồi nhanh và dễ bảo trì thông qua cơ chế component hóa và các hook như useState, useEffect.

Đồng thời, sử dụng Tailwind CSS để thiết kế giao diện nhanh chóng và đồng bộ bằng các lớp tiện ích trực tiếp trong JSX.

Công nghệ sử dụng phát triển Back-end

Phát triển Backend với Spring Boot làm nền tảng chính nhờ tính linh hoạt, dễ cấu hình và khả năng mở rộng cao.

Tích hợp thêm một số thư viện và công cụ hỗ trợ mạnh mẽ cho việc xây dựng, bảo mật, xử lý dữ liệu và kiểm thử ứng dụng: OAuth2 Resource Server, Springdoc OpenAPI (Swagger UI), Validation, Spring Security Crypto, Lombok, MapStruct, JaCoCo Maven Plugin, H2 Database và Spring Security Test

Công nghệ cơ sở dữ liệu

Sử dụng MySQL 8.0 làm hệ quản trị cơ sở dữ liệu, triển khai qua Docker để dễ tích hợp với hệ thống. DBeaver hỗ trợ quản lý và truy vấn dữ liệu trong quá trình phát triển. Spring Data JPA được dùng để tương tác với cơ sở dữ liệu qua các đối tượng Java, giúp truy xuất dữ liệu hiệu quả mà không cần viết SQL thủ công.

2. QUY TRÌNH CI/CD VỚI GITHUB ACTION

GitHub Actions, nhằm tự động hóa việc kiểm tra, xây dựng và triển khai hệ thống backend mỗi khi có thay đổi trong mã nguồn.

Cụ thể, mỗi khi có thao tác push code hoặc pull request lên GitHub, một workflow Super Linter được kích hoạt tự động để thực hiện các bước sau:

- + Kiểm tra cú pháp và định dạng của các tệp Java và JSX trong mã nguồn.
- + Phát hiện các lỗi phổ biến hoặc đoạn mã không tuân theo quy chuẩn.
- + Đảm bảo tính nhất quán và sạch sẽ của mã trước khi được merge vào nhánh chính.

Như vậy có thể phát hiện sớm lỗi và duy trì chất lượng mã một cách tự động, liên tục trong quá trình phát triển.

2. CẤU HÌNH DOCKER VÀ TRIỂN KHAI ỨNG DỤNG

Cấu hình Docker

Hệ thống ứng dụng được container hóa với Docker để đảm bảo ứng dụng có thể chạy ổn định, nhất quán trên mọi môi trường. Hệ thống được chia thành ba container chính, mỗi container đảm nhận một vai trò riêng biệt:

- Frontend (ReactJS): build và khởi chạy React SPA bằng Node.js và được phục vụ với Nginx trong container với cổng EXPOSE: 80.

- Backend (Spring Boot): được build trong container Maven sử dụng JDK 21, sau đó đóng gói backend (tệp .jar) rồi được chuyển sang một container nhẹ chạy trên Amazon Corretto 21. Ứng dụng sẽ được khởi chạy trên cổng 8386.

Triển khai ứng dụng

Sau khi đóng gói ứng dụng thành các container, sử dụng Docker Compose để Dockerize ứng dụng và quản lý toàn bộ hệ thống một cách dễ dàng và nhất quán. File docker-compose.yml định nghĩa các dịch vụ chính là backend, frontend và cơ sở dữ liệu. Với các cổng phụ thuộc (depends_on) như backend kết nối với cơ sở dữ liệu. Sau khi Dockerize ứng dụng, để build và khởi động các container chạy lệnh docker-compose up -build.

Khi build thành công, ứng dụng được chạy cục bộ trên máy chủ tại địa chỉ localhost:3000, cho phép truy cập giao diện người dùng. Đồng thời ứng dụng cũng sẽ có thể tương tác với các chức năng hệ thống qua giao diện như: đăng ký/đăng nhập, đặt bàn, xem thông tin, ...

KIỂM THỬ

Kiểm thử API với Postman

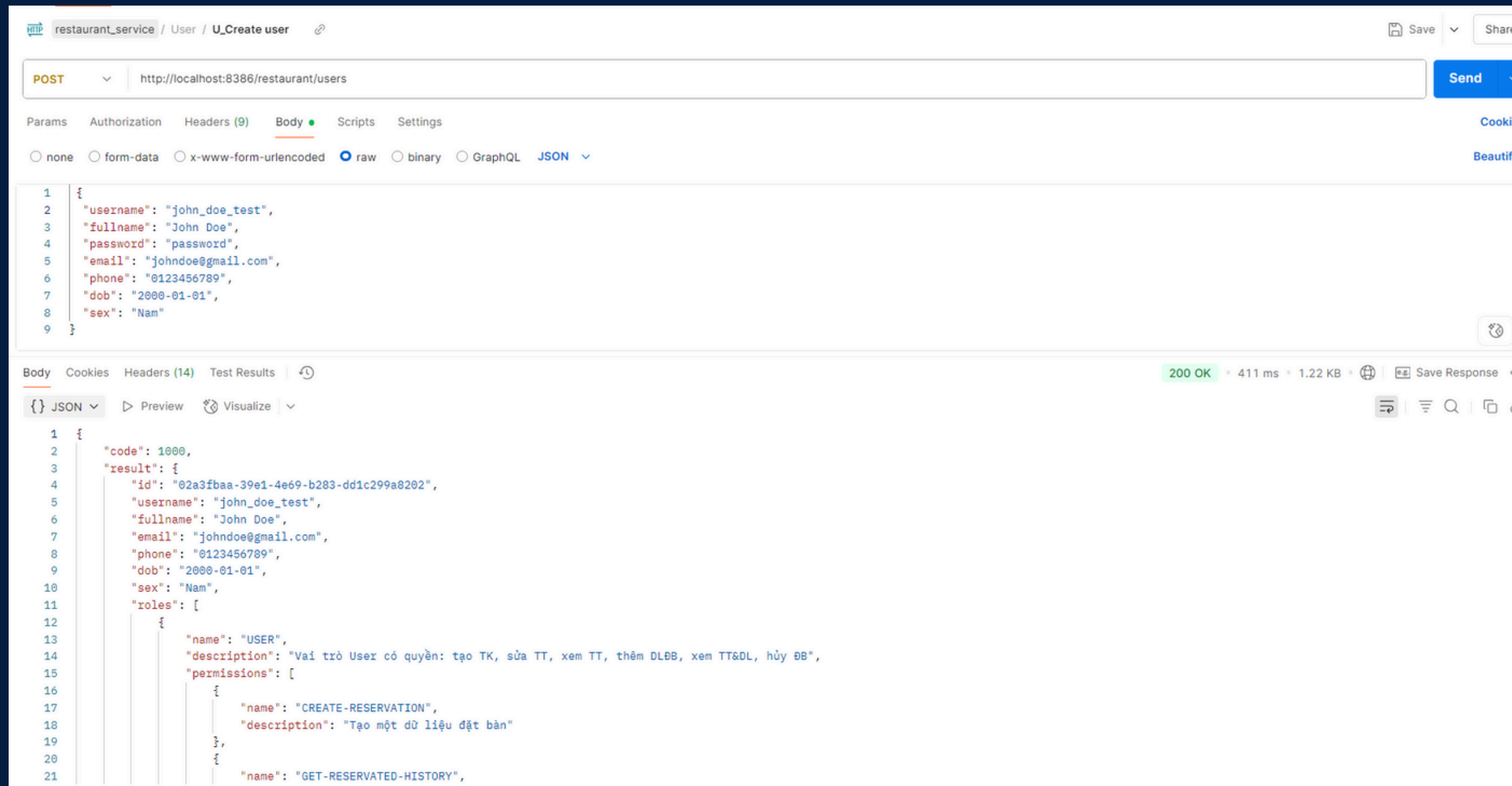
Unit test

Coverage với Jacoco

```
UserCreationRequest request  
Username(request.getUserName)  
ErrorCode.USER_EXISTED);  
  
er(request);  
coder.encode(request.getPassword)  
  
ull || request.getRoles().isEmpty(),  
epository.findByName("USER")  
new ApplicationException(ErrorCode.ROLE_NOT  
ns.singleton(defaultRole));  
  
ory.findAllById(request.getRol  
et<>(roles));  
  
ponse(userRepository.save
```

KIỂM THỬ API VỚI POSTMAN

Dùng Postman để kiểm thử hiệu quả các API backend bằng cách gửi các yêu cầu HTTP (GET, POST, PUT, DELETE) đến các endpoint chức năng như đăng ký, đăng nhập, đặt bàn, quản lý người dùng,... Việc này giúp xác minh tính đúng đắn, phản hồi dữ liệu (JSON), mã trạng thái và thời gian xử lý. Postman cũng hỗ trợ kiểm tra xác thực bằng cách gửi JWT token trong headers để đảm bảo bảo mật và phân quyền cho các API.



UNIT TEST

Viết Unit test cho lớp controller và service nhằm đảm bảo các API hoạt động đúng với yêu cầu đầu vào và đầu ra.









Sử dụng JUnit 5, Mockito, Spring Boot Test kết hợp với MockMvc để xây dựng và thực thi các test case, mô phỏng các yêu cầu HTTP và kiểm tra phản hồi từ các endpoint RESTful.

Controller được kiểm thử với các kịch bản hợp lệ và không hợp lệ: kiểm tra mapping, định dạng request/response, và gọi đúng service. Dùng @MockBean để mock service, giúp cô lập logic controller.

Service được kiểm thử bằng cách mock các thành phần phụ thuộc như UserRepository, đảm bảo logic xử lý đúng như kiểm tra dữ liệu, lưu DB, xử lý lỗi, mã hóa mật khẩu và tạo đối tượng phản hồi.

COVERAGE VỚI JACOCO

Đo lường tỷ lệ bao phủ mã nguồn của test case với Jacoco (Java Code Coverage), giúp xác định những phần nào của mã nguồn đã được kiểm thử và phần nào chưa. Đối với hệ thống này đã được viết Unit test với tỷ lệ bao phủ cao trên 70%.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 com.option1.restaurant_service.service	<div><div></div></div>	54%	<div><div></div></div>	20%	18	46	73	141	9	36	0	4
 com.option1.restaurant_service.service.restaurantService	<div><div></div></div>	80%	<div><div></div></div>	57%	15	47	18	113	12	40	0	3
 com.option1.restaurant_service.exception	<div><div></div></div>	86%	<div><div></div></div>	50%	4	16	16	75	3	15	0	3
 com.option1.restaurant_service.controller	<div><div></div></div>	78%		n/a	4	24	12	57	4	24	0	4
 com.option1.restaurant_service.controller.RestaurantController	<div><div></div></div>	85%		n/a	3	22	9	54	3	22	0	3
 com.option1.restaurant_service.validation	<div><div></div></div>	90%	<div><div></div></div>	50%	2	5	1	8	0	3	0	1
 com.option1.restaurant_service.enums	<div><div></div></div>	76%		n/a	1	4	3	13	1	4	1	2
 com.option1.restaurant_service		37%		n/a	1	2	2	3	1	2	0	1
Total	537 of 2.032	73%	25 of 40	37%	48	166	134	464	33	146	1	21