Nexlab Technology

Backend Internship Test

Câu 1. Database:

a. MongoDB là gì?
MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL dạng document-oriented, cho phép lưu trữ dữ liệu dưới dạng JSON-like (BSON). MongoDB có khả năng mở rộng cao và linh hoạt, phù hợp với dữ liệu không có schema cố định và thay đổi thường xuyên.

b. PostgreSQL là gì?
PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) hỗ trợ ACID, sử dụng SQL để truy vấn dữ liệu. PostgreSQL thích hợp cho các ứng dụng cần tính nhất quán cao.

c. ORM là gì?
ORM (Object-Relational Mapping) là kỹ thuật ánh xạ các bảng trong cơ sở dữ liệu quan hệ sang đối tượng trong lập trình hướng đối tượng, giúp thao tác với database bằng code thay vì viết SQL trực tiếp.

d. Database migration là gì?
Database migration là quá trình quản lý thay đổi schema database (thêm/xóa/sửa bảng, cột, ràng buộc) một cách có hệ thống, đảm bảo đồng bộ giữa các môi trường phát triển, staging và production. Với PostgreSQL, TypeORM hỗ trợ migration tự động.

e. Liệt kê một số ORM phổ biến của MongoDB và PostgreSQL.
- **MongoDB:** Mongoose.
- **PostgreSQL:** TypeORM, Sequelize, Prisma.

f. So sánh MongoDB và PostgreSQL.

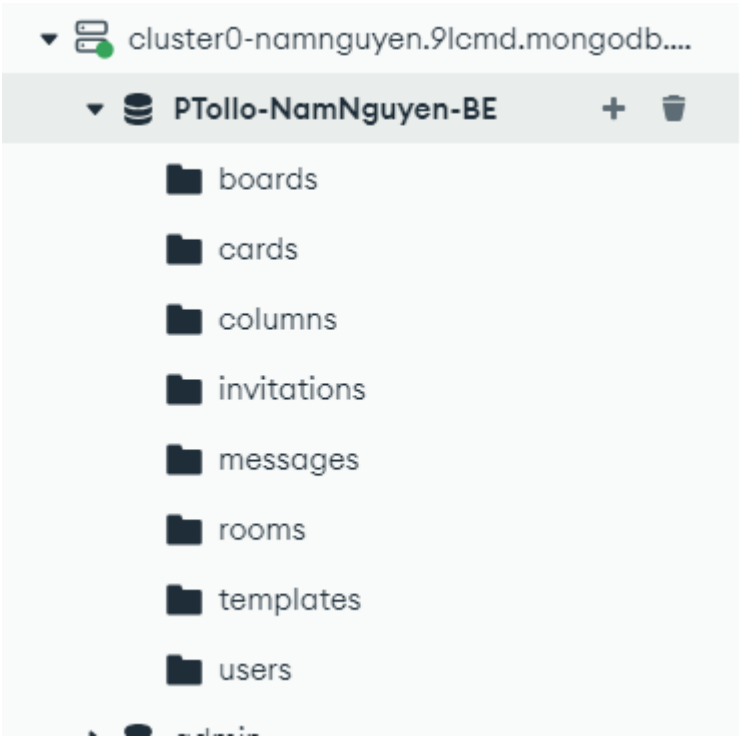
Tiêu chí	MongoDB (NoSQL)	PostgreSQL (SQL)
Schema	Linh hoạt, không cần định nghĩa trước	Cố định, yêu cầu schema rõ ràng
Scalability	Horizontal scaling (sharding) Tốt với dữ liệu lớn	Vertical scaling, hỗ trợ replication Phù hợp với giao dịch phức tạp
Use Case	Dữ liệu phi cấu trúc, real-time Nhanh hơn	Giao dịch phức tạp, báo cáo Chậm hơn nếu dữ liệu lớn
ACID	Hỗ trợ từ phiên bản 4.0	Hỗ trợ đầy đủ

g. Bạn đã từng làm việc với hai database trên chưa? Bạn đã từng sử dụng ORM và migration chưa? Nếu có, hãy nêu ra một số kinh nghiệm thực tế.

Em đã có kinh nghiệm làm việc với MongoDB và sử dụng ORM Mongoose để quản lý dữ liệu trong dự án gần đây của mình – một ứng dụng quản lý dự án cá nhân và nhóm (Ptollo). Trong dự án này, em sử dụng MongoDB để lưu trữ dữ liệu một cách linh hoạt, bao gồm các thông tin quan trọng như:

- **Boards: Quản lý các dự án.**
- **Cards & Columns: Lưu trữ công việc và danh sách công việc trong từng dự án.**
- **Invitations: Hệ thống mời thành viên tham gia dự án.**

- **Messages: Lưu trữ tin nhắn trong hệ thống chat.**
- **Room Call Group: Hỗ trợ tính năng video call nhóm.**
- **User Management: Quản lý thông tin người dùng.**



MongoDB tỏ ra rất phù hợp với dự án này do tính linh hoạt của **NoSQL**, giúp em có thể lưu trữ dữ liệu không cần schema cố định, dễ dàng mở rộng ứng dụng hơn.

Về PostgreSQL, em mới tìm hiểu gần đây nhưng đã có kinh nghiệm làm việc với các hệ quản trị cơ sở dữ liệu quan hệ (SQL-based DBMS) như MySQL và SQL Server trong các dự án trước đây. Một trong số đó là dự án lập trình game socket Battle Ship, nơi em sử dụng SQL Server để lưu trữ dữ liệu như:

- **User Data: Thông tin tài khoản người chơi.**
- **Room & Lobby: Quản lý các phòng chơi và sảnh chờ.**
- **Game History: Lưu trữ kết quả các trận đấu.**

Trong dự án này, em không sử dụng ORM mà viết **các truy vấn SQL trực tiếp** để thao tác với database, giúp tối ưu hiệu suất khi xử lý dữ liệu thời gian thực cho game.

Em nhận thấy rằng PostgreSQL có nhiều điểm tương đồng với MySQL và SQL Server, đặc biệt là trong cách thiết kế **schema, query SQL và quản lý ràng buộc dữ liệu**. Tuy chưa có cơ hội triển khai PostgreSQL trong dự án thực tế, nhưng em cũng kinh nghiệm làm việc với các hệ quản trị SQL trước đó.

Câu 2. RESTful API:

a. RESTful API là gì?
RESTful API là một kiểu thiết kế API tuân thủ nguyên tắc REST (Representational State Transfer), sử dụng các phương thức HTTP (GET, POST, PUT, DELETE) để thao tác với tài nguyên thông qua các endpoint.

b. Liệt kê một số phương thức HTTP trong RESTful.
   - **GET: Lấy dữ liệu.**
   - **POST: Tạo mới.**

- **PUT: Cập nhật toàn bộ.**
- **PATCH: Cập nhật một phần.**
- **DELETE: Xóa.**

c. Bạn đã làm việc với RESTful API bao giờ chưa? Nếu có, hãy nêu ra một số kinh nghiệm thực tế.

Em đã có kinh nghiệm làm việc với **RESTful API** trong dự án **quản lý dự án và nhóm (Ptollo)**. Trong dự án này, em đã thiết kế và triển khai nhiều API theo kiến trúc **RESTfull**, sử dụng **Express.js và MongoDB** để xây dựng hệ thống quản lý bảng làm việc (boards), danh sách công việc (cards, columns) và các tính năng mở rộng như gắn sao, cập nhật trạng thái hoàn thành, di chuyển thẻ công việc.

Dưới đây là một số API quan trọng mà em đã triển khai trong hệ thống:

- **GET /v1/boards: Lấy danh sách các bảng mà người dùng có quyền truy cập.**
- **POST /v1/boards: Tạo một bảng mới, yêu cầu xác thực người dùng.**
- **GET /v1/boards/: id: Lấy thông tin chi tiết của một bảng làm việc.**
- **PUT /v1/boards/: id: Cập nhật thông tin của bảng (tên, mô tả...).**
- **POST /v1/boards/: id/recently_viewed: Đánh dấu bảng là đã xem gần đây.**
- **PUT /v1/boards/: id/star_board: Gắn sao một bảng để đánh dấu quan trọng.**
- **PUT /v1/boards/supports/moving_card: Cho phép người dùng di chuyển một card từ danh sách này sang danh sách khác trong bảng.**
- **PUT /v1/boards/: id/background: Cập nhật màu nền hoặc ảnh nền của bảng.**
- **PUT /v1/boards/: id/upload-image: Upload ảnh nền mới cho bảng.**
- **GET /v1/boards/: id/completion-board: Tính toán và trả về mức độ hoàn thành của bảng, dựa trên tiến độ của các công việc bên trong.**
- **POST /v1/boards/: id/template: Chuyển một bảng làm việc thành template có thể tái sử dụng cho các dự án khác.**

Tổng kết:

- Em đã triển khai API theo **RESTful standards**, đảm bảo tính nhất quán và dễ mở rộng.
- Áp dụng **Middleware (authMiddleware)** để xác thực người dùng trước khi họ truy cập hoặc thay đổi dữ liệu.
- Sử dụng **Validation (boardValidation)** để kiểm tra dữ liệu đầu vào, giúp giảm lỗi khi gửi request.
- Hỗ trợ **file upload** cho tính năng thay đổi ảnh nền của bảng.
- Xây dựng API hỗ trợ **tương tác real-time**, giúp người dùng cập nhật tiến độ công việc một cách trực quan.
- Đảm bảo API tối ưu, giúp hệ thống hoạt động ổn định ngay cả khi số lượng người dùng tăng cao.

Github: https://github.com/UIT-20521633/FullStack-Ptollo.git

Câu 3. API document:

a. API document là gì?
Là tài liệu mô tả cách sử dụng API, bao gồm endpoints, parameters, request/response format, status codes...

b. Liệt kê một số công cụ để làm API document phổ biến.
- Postman (test API)
- Swagger (tạo tài liệu API tự động)

c. Bạn đã làm việc với API document bao giờ chưa? Nếu có, hãy nêu ra một số kinh nghiệm thực tế.
Em đã sử dụng Postman để kiểm thử các API trong dự án Back-end của mình. Postman giúp em dễ dàng gửi yêu cầu (request) đến API, kiểm tra phản hồi (response) và mô phỏng các luồng xử lý trong hệ thống.
Ngoài ra, em cũng sử dụng Postman Collections để tổ chức và chia sẻ API, giúp team dễ dàng kiểm thử và tích hợp hệ thống. Đây là public link bộ sưu tập API của em trên
Postman: https://api-team-5524.postman.co/workspace/My-Workspace~2952d39a-fcc8-48ac-8652-2f319422467c/collection/39955083-50150542-8a6e-4425-9df0-27c95212c3bf?action=share&creator=39955083

Các bước em đã thực hiện với Postman:
- Tạo và kiểm thử các API quan trọng: Đăng ký, đăng nhập, quản lý bảng, di chuyển thẻ, cập nhật thông tin.
- Lưu và chia sẻ Collection API để đảm bảo tài liệu API luôn đồng bộ với team.
- Mock API với Postman để kiểm tra phản hồi trước khi triển khai thực tế.

Câu 4. Git:

a. Git là gì?
Hệ thống quản lý phiên bản phân tán, giúp theo dõi thay đổi mã nguồn và hợp tác nhóm.

b. Hãy liệt kê một số lệnh Git cơ bản?
git init – Khởi tạo repo
git clone – Sao chép repo
git add . – Thêm file vào staging
git commit -m 'message' – Lưu thay đổi
git push origin main – Đẩy code lên server
git merge <branch>: Gộp nhánh vào branch chính.
git branch: Quản lý nhánh
git pull origin <branch>: Lấy code mới nhất từ remote repository
git checkout -b <branch>: Tạo và chuyển sang một nhánh mới.

c. Bạn đã từng làm việc với Git chưa? Nếu có, hãy nêu ra một số kinh nghiệm thực tế.

Hầu như tất cả các dự án của em đều sử dụng **Git** để quản lý mã nguồn. Git giúp em theo dõi lịch sử thay đổi của mã nguồn, phối hợp làm việc nhóm và đảm bảo tính nhất quán của dự án.

Em sử dụng GitHub để lưu trữ và quản lý dự án của mình. Đây là public repository của em trên

GitHub: https://github.com/UIT-20521633

Cách em sử dụng Git trong dự án:

- Khởi tạo và quản lý repository: Clone một repository mới khi bắt đầu dự án.
- Tạo nhánh (branch) riêng biệt cho từng tính năng: Mỗi thành viên trong team làm việc trên nhánh riêng để phát triển từng module cụ thể.
- Code review và merge vào branch chính: Sử dụng pull request để kiểm tra code trước khi hợp nhất vào main hoặc develop.
- Sử dụng Git để làm việc nhóm: Push, pull code thường xuyên để đảm bảo tất cả thành viên trong nhóm đều cập nhật mã nguồn mới nhất.

Câu 5. SQL query (sử dụng PostgreSQL):

a. **Cho hai table: visit và transaction. Tìm id của những người đã truy cập mà không thực hiện bất kỳ giao dịch nào và số lần họ thực hiện các kiểu truy cập này.**

visit visit_id: int customer_id: int primary_key: visit_id	**transaction** transaction_id: int visit_id: int amount: int primary_key: transaction_id

SQL Query:

```
SELECT v.customer_id, COUNT(v.visit_id) AS visit_count
FROM visit v
LEFT JOIN transaction t ON v.visit_id = t.visit_id
WHERE t.transaction_id IS NULL
GROUP BY v.customer_id;
```

b. Cho table log. Tìm tất cả các số xuất hiện liên tiếp ít nhất ba lần

log
id: int
num: text
primary_key: id

Ví dụ:

id	num
1	1
2	1
3	1
4	2
5	1
6	2
7	2

SQL Query

```
WITH temp AS (
  SELECT
    num,
    LAG(num, 1) OVER (ORDER BY id) AS prev1,
    LAG(num, 2) OVER (ORDER BY id) AS prev2
  FROM log
)
SELECT DISTINCT num
FROM temp
WHERE num = prev1 AND num = prev2;
```

c. Cho table activity. Viết giải pháp báo cáo tỷ lệ người chơi đăng nhập lại vào ngày tiếp theo sau ngày đăng nhập lần đầu.

Gợi ý: Đếm số lượng người chơi đã đăng nhập trong ít nhất hai ngày liên tiếp bắt đầu từ ngày đăng nhập đầu tiên, sau đó chia số đó cho tổng số người chơi.

activity
player_id: int
device_id: int
event_date: date
games_played: int
primary_key: player_id + event_date

SQL Query

```
WITH first_login AS (
  SELECT
    player_id,
    MIN(event_date) AS first_date
  FROM activity
  GROUP BY player_id
)
SELECT
  ROUND(
    COUNT(DISTINCT a.player_id) * 1.0 /
    (SELECT COUNT(DISTINCT player_id) FROM activity),
  2
) AS retention_rate
FROM activity a
JOIN first_login f ON a.player_id = f.player_id
WHERE a.event_date = f.first_date + INTERVAL '1 day';
```

Câu 7. Design System & Code:

Mô tả hệ thống Quản lý cửa hàng:

- Chủ cửa hàng có thể đăng ký tài khoản để quản lý các cửa hàng của mình. Thông tin của một chủ cửa hàng bao gồm: họ và tên, số điện thoại, email, ngày tháng năm sinh, ảnh đại diện, giới tính, và một mã định danh unique được generate tự động trên hệ thống.
- Mỗi chủ cửa hàng có thể có một hoặc nhiều cửa hàng, thông tin cửa hàng: tên, logo, địa chỉ (bao gồm số nhà, đường, phường, quận và tỉnh thành), số điện thoại, email.
- Trong hệ thống sẽ có các người làm tự do có thể đăng ký tài khoản trên hệ thống và có các thông tin như: họ và tên, số điện thoại (không được trùng), email, ngày tháng năm sinh, giới tính, ảnh đại diện, địa chỉ (bao gồm số nhà, đường, phường, quận và tỉnh thành).
- Những người làm tự do có thể gửi yêu cầu làm việc cho một cửa hàng bằng cách tìm kiếm danh sách cửa hàng thông qua mã định danh của chủ cửa hàng. Sau khi chọn được cửa hàng phù hợp thì sẽ gửi yêu cầu làm việc đến cho chủ cửa hàng và đợi chủ cửa hàng duyệt yêu cầu. Ngoài ra, chủ cửa hàng cũng có thể tìm kiếm người làm tự do thông qua số điện thoại, sau đó chọn cửa hàng và gửi đề nghị làm việc và đợi cho người làm này phản hồi đề nghị. Tại một thời điểm, mỗi người làm tự do chỉ được làm cho một cửa hàng duy nhất, chủ cửa hàng chỉ có thể gửi đề nghị đến những người làm đang không có hợp đồng làm việc.
- Chủ cửa hàng có thể xem danh sách các cửa hàng và danh sách nhân viên đang làm việc cho từng cửa hàng. Chủ cửa hàng có thể đơn phương chấm dứt hợp đồng với một nhân viên bất kỳ.
- Chủ cửa hàng có thể xem được danh sách yêu cầu làm việc từ các người làm tự do, và phản hồi chấp nhận hoặc từ chối các yêu cầu đó.
- Người làm tự do có thể xem yêu cầu làm việc hoặc đề nghị, phản hồi từ chối hoặc chấp nhận đề nghị.
- Sau khi đăng ký tài khoản, người dùng cần xác thực để kích hoạt tài khoản thông qua email. Chỉ các tài khoản đã xác thực mới có thể đăng nhập vào hệ thống.
- Chủ cửa hàng và người làm tự do có thể sử dụng email/số điện thoại và mật khẩu (được mã hóa trước khi lưu vào hệ thống) để đăng nhập vào hệ thống.

a. Hãy thiết kế database cho hệ thống này. Sử dụng draw.io, public link để nộp bài.

Link bài: https://drive.google.com/file/d/1fxbOEIgODZNp0nj2rrUZk6Gq_fSa8U-C/view?usp=sharing

b. Hãy liệt kê và nêu mục đích các APIs cần thiết để quản lý hệ thống trên

1. API Người dùng (User APIs)

Xác thực & Quản lý tài khoản

- **POST** /api/auth/register - Đăng ký tài khoản (chủ cửa hàng hoặc người làm tự do).
- **POST** /api/auth/login - Đăng nhập vào hệ thống.
- **POST** /api/auth/verify-email - Xác thực email để kích hoạt tài khoản.
- **GET** /api/auth/refresh_token - Cấp lại token xác thực.
- **POST** /api/auth/forgot-password - Gửi email để đặt lại mật khẩu.
- **POST** /api/auth/reset-password - Đặt lại mật khẩu.

2. API Chủ cửa hàng & Cửa hàng (StoreOwner & Store APIs)

Quản lý Chủ cửa hàng

- **PUT** /api/owner - Cập nhật thông tin cá nhân của chủ cửa hàng.
- **GET** /api/owners/stores - Lấy danh sách cửa hàng của chủ cửa hàng.
- **GET /api/owners/stores/: storeId/employees - Lấy danh sách nhân viên của một cửa hàng.**
- **DELETE /api/owners/contracts/: contractId - Chấm dứt hợp đồng với nhân viên.**

Quản lý Cửa hàng

- **POST** /api/stores - Tạo cửa hàng mới.
- **PUT /api/stores/: storeId - Cập nhật thông tin cửa hàng.**
- **DELETE /api/stores/: storeId - Xóa cửa hàng khỏi hệ thống.**

3. API Người làm tự do (Freelancer APIs)

Quản lý tài khoản Freelancer

- **PUT** /api/freelancers - Cập nhật thông tin cá nhân của người làm tự do.

Tìm kiếm & Ứng tuyển công việc

- **GET** /api/freelancers/search-stores - Tìm kiếm cửa hàng bằng mã định danh chủ cửa hàng.
- **POST** /api/freelancers/job-requests - Gửi yêu cầu làm việc đến cửa hàng.
- **GET** /api/freelancers/job-offers - Lấy danh sách đề nghị làm việc.
- **PUT /api/freelancers/job-offers/: offerId - Phản hồi đề nghị làm việc (chấp nhận/từ chối).**

4. API Yêu cầu & Đề nghị công việc (JobRequest & JobOffer APIs)

Quản lý Yêu cầu công việc

- **GET** /api/owners/job-requests - Lấy danh sách yêu cầu làm việc từ người làm tự do.
- **PUT /api/owners/job-requests/: requestId - Phản hồi yêu cầu làm việc (chấp nhận/từ chối).**

Quản lý Đề nghị công việc

- **POST** /api/owners/job-offers - Gửi đề nghị làm việc đến người làm tự do.

c. Chọn ra ít nhất 3 APIs đã liệt kê ở trên để viết code.
Yêu cầu kỹ thuật:

- **Framework: NestJS/ ExpressJS.**
- **Database: PostgreSQL.**
- **ORM & Migration: TypeORM.**
- **API Document: Postman (public link để nộp bài có lưu example request & response cho từng API).**
- **GIT: sử dụng github/ gitlab/ bitbucket/ ... để quản lý source code (public link để nộp bài).**
- **Demo: quay video demo các API trên upload lên Youtube, public link để nộp bài.**

API Doccument:
Link1 :
https://api-team-5524.postman.co/workspace/My-Workspace~2952d39a-fcc8-48ac-8652-2f319422467c/collection/39955083-ef5bfa4f-346e-4716-b630-795cccc35780?action=share&creator=39955083

Link dự phòng:
https://drive.google.com/file/d/1gykWx_AnGSfz6HFCfzWUge6RSRt-a8Na/view?usp=sharing

GIT:
https://github.com/UIT-20521633/APIQLCH-NexLab.git

Demo:
https: //youtu.be/U8tKOH4jDUM

Chúc các bạn hoàn thành bài kiểm tra thật tốt!!!