LỜI CẢM ƠN

Em xin gửi đến cô Ngô Thị Duyên, người đã hướng dẫn cho em để có thể hoàn thành bộ môn thực tập công ty này lời cảm ơn sâu sắc nhất.

Em cũng gửi lời cảm ơn tới công ty CMC Global vì sự giúp đỡ và tạo điều kiện thuận lợi cho em về mọi mặt, tìm hiểu thực tiễn và hướng dẫn trong suốt quá trình làm việc tại công ty. Trong quá trình thực tập khó tránh khỏi những sai sót, em mong được cô bỏ qua. Em xin chân thành cảm ơn!

1. Giới thiệu chung
   1. Giới thiệu công ty

CMC Global, thành viên của Tập đoàn Công nghệ CMC được thành lập vào ngày 31/03/2017. CMC Global được thành lập với tầm nhìn trở thành công ty dịch vụ Top 10 trong khu vực, và là đơn vị đứng đầu về năng lực triển khai các giải pháp công nghệ mới như AI, Cloud và chuyển đổi số và sứ mệnh “tạo ra dịch vụ đạt chuẩn quốc tế, hiện diện trên toàn cầu”. Với tầm nhìn, sứ mệnh đó, CMC Global tự hào thuộc TOP 2 công ty IT Outsourcing tại Việt Nam hiện nay, với khát vọng đem những sản phẩm, giải pháp và dịch vụ CNTT Việt Nam ra thị trường quốc tế.

Với 5 năm phát triển thần tốc, từ 134 thành nhân sự khi công ty thành lập vào năm 2017, đến nay, công ty đã phát triển thần tốc với số lượng nhân sự vượt mức 2200 nhân viên. Và mục tiêu đến năm 2025, CMC Global sẽ có 7000 nhân sự làm việc tại công ty.

* 1. Giới thiệu công việc

Trong thời gian làm việc ở công ty, em được học tập nhiều kiến thức về lập trình nói chung và đi sâu hơn về lập trình ứng dụng web. Công việc chính mà em thực hiện trong giai đoạn này đó là phát triển các tính năng mới theo yêu cầu của khách hàng, sửa các lỗi còn tồn đọng trong dự án.

1. Yêu cầu chất lượng
   1. Ngôn ngữ, framework sử dụng

* Frontend sử dụng ReactJS, backend dùng NodeJS.
  1. Yêu cầu chất lượng
* Các trang giao diện người dùng phải minh bạch, rõ ràng theo đúng như thiết kế trên Figma.
* Các component ReactJS phải tách biệt chức năng, áp dụng được nguyên lí SOLID
* Quản lí các state của component để hạn chế việc re-render quá nhiều, giảm hiệu năng ứng dụng
* Xử lí caching lại data từ server

1. Tóm tắt lý thuyết, giải pháp, thuật toán.
   1. Mô tả chung về kiến trúc và xây dựng hệ thống

Hệ thống gồm 2 phần chính là frontend và backend:

* Frontend sử dụng ReactJS để tạo nên phần giao diện người dùng, bao gồm các component
* Backend được viết bằng NodeJS, sử dụng các dịch vụ của AWS như Lambda và Step Function để xây dựng nên hệ thống backend serverless.
  1. Một số lý thuyết, công cụ sử dụng và dịch vụ nổi bật được ứng dụng và cách giải quyết vấn đề trong hệ thống
* React Query: React Query là một thư dành cho React, giúp quản lý việc lấy, lưu trữ và đồng bộ hóa dữ liệu từ máy chủ. Nó giúp đơn giản hóa các thao tác với dữ liệu không đồng bộ, thay thế cho cách quản lý local state phức tạp trong các ứng dụng React. 1 vài tính năng chính có thể kể đến như:
  + Quản lý cache tự động: Tự động lưu trữ và quản lý dữ liệu đã lấy, giảm thiểu số lần gọi API không cần thiết.
  + Xử lý trạng thái yêu cầu: Cung cấp các trạng thái như loading, error, success giúp dễ dàng quản lý và hiển thị UI phù hợp.
  + Refetching và Background Sync: Tự động cập nhật dữ liệu khi có thay đổi hoặc theo khoảng thời gian định kỳ.

Trong ứng dụng em làm, phía giao diện sẽ có 1 bảng gồm nhiều bản ghi. Mỗi bản ghi có 1 ô cần phải lấy dữ liệu từ server về. Do có nhiều bản ghi nên mỗi bản ghi lại gọi 1 lần API lên server gây ra trùng lặp, tốn băng thông và bộ nhớ của ứng dụng. Vậy nên em đã sử dụng React Query để cache lại dữ liệu sau khi lấy về, các bản ghi khác chỉ việc lấy từ cache ra mà không phải gọi API lại nữa.

* Zustand: Đây là một thư viện quản lý state dành cho React, nổi bật với sự đơn giản và hiệu suất cao. Nó giúp em dễ dàng quản lý và chia sẻ state trong ứng dụng mà không cần phải làm việc với boilerplate code phức tạp.

1 ví dụ nhỏ: trong ứng dụng khi em xây dựng tính năng bộ lọc data (filter). Bộ lọc này gồm nhiều tiêu chí khác nhau. Nhưng các bản ghi và bộ lọc được đặt ở các component khác nhau. Nếu không có cách chia sẻ state (ở đây là các bộ lọc) thì sẽ phải khai báo các state này ở component cha gần nhất, và truyền dần các state này xuống nơi cần dùng chúng. Điều này dẫn đến việc tham số truyền vào cho component rất khó quản lí, đặc biệt khi các tiêu chí lọc ngày càng nhiều lên. Thêm vào đó, một khi component cha rerender, tất cả các component con được truyền tham số xuống đều sẽ rerender tạo nên hiệu ứng domino gây tốn kém về mặt hiệu năng. Vì vậy mà em sử dụng zustand để chia sẻ các state tiêu chí này. Khi cần dùng chỉ việc import vào component và lấy ra mà không phải truyền qua tham số.

* Compound Component: Compound components là một kỹ thuật trong React giúp xây dựng các thành phần giao diện linh hoạt, cho phép các thành phần con giao tiếp với nhau và chia sẻ logic một cách tự nhiên. Nó còn giúp em customize các component con tuỳ ý.

Trong ứng dụng, em sử dụng compound component để xây dựng 1 bảng gồm nhiều các bản ghi. Bảng này sẽ gồm nhiều component như Header, Cells, Column, Pagination. Nếu như em muốn dùng cái bảng này trong 1 trang giao diện khác, mà không có nhu cầu thêm Header hoặc Pagination thì sẽ đơn giản chỉ là không thêm 2 thẻ này vào thẻ Table ở trên thôi.

* useMemo: Đây là một hook trong React giúp tối ưu hóa hiệu suất bằng cách ghi nhớ giá trị đã tính toán lại giữa các lần render. Nó chỉ tính toán lại giá trị khi các dependencies thay đổi, giúp tránh các tính toán không cần thiết. useMemo nhận hai tham số: 1 là hàm tính toán giá trị cần ghi nhớ, 2 là mảng các giá trị mà khi thay đổi sẽ khiến useMemo tính toán lại

Quay lại với bộ lọc filter gồm nhiều các tiêu chí bên trên, giả sử có hàng nghìn bản ghi và 10 tiêu chí. Mỗi khi lọc thì 1 mảng các bản ghi sẽ cần phải được duyệt qua 10 lần để lấy ra kết quả. Nếu như có 1 tiêu chí thay đổi thì mảng sẽ phải chạy lại qua cả 10 lần nữa. Việc này gây ra tốn kém hiệu năng. Em xử lí bằng cách mỗi khi mảng được lọc qua 1 tiêu chí, em cho đoạn code xử lí việc lọc vào tham số thứ nhất của useMemo, và cho bản ghi và tiêu chí đó vào tham số thứ hai. Bằng cách này, chỉ khi bản ghi hoặc tiêu chí thay đổi thì hàm mới chạy lại, còn nếu các tiêu chí khác thay đổi thì kết quả của hàm lọc vẫn được giữ nguyên.

* GraphQL Subscription: Trong ứng dụng, khi em bấm 1 nút bấm, phía client sẽ gọi đến và kích hoạt 1 AWS Step Functions. 1 Step Functions sẽ gồm nhiều lambda khác nhau và cần có thời gian để nó chạy hết. Nhưng vấn đề là sau khi bấm phía client không biết trước được khi nào thì Step Functions chạy xong để update lại giao diện tương ứng. Do đó sẽ có lỗi nếu người dùng bấm 2 lần nút đó thì sẽ gọi 2 lần Step Functions. Để giải quyết điều này thì em dùng Subscription của GraphQL.
* GraphQL Subscriptions là một tính năng cho phép server gửi dữ liệu thời gian thực đến client khi có sự kiện xảy ra. Điều này rất hữu ích cho các ứng dụng cần cập nhật liên tục, như chat, thông báo, hay theo dõi trạng thái. Sau khi bấm nút, em sẽ disabled lại và thể hiện trạng thái loading bằng css. Sau đó, khi Step Function chạy xong, em sẽ thêm 1 bản ghi vào 1 bảng có tác dụng như lịch sử các thông báo, và ở phía Client GraphQL sẽ theo dõi bảng này. Mỗi khi có bản ghi mới được thêm vào, GraphQL sẽ thông báo lại client. Lúc này em sẽ gọi lại API để lấy data mới về và cập nhật lại dữ liệu

1. Kết quả đạt được, hướng phát triển
   1. Kỹ năng & kiến thức đã học được

Sau 2 tháng học và làm việc tại công ty, em đã thu thập được rất nhiều kiến thức. Dưới đây là vài mục nổi bật:

* Có khả năng tự tìm hiểu và giải quyết các vấn đề
* Có khả năng teamwork và làm việc trong dự án sử dụng Scrum
* Áp dụng các cấu trúc dữ liệu và giải thuật vào các vấn đề thực tế
* Hiểu về Coding Convention
* Hiểu được một số design pattern cơ bản thường dùng
* Biết và áp dụng Git flow trong quá trình phát triển tính năng
* Biết cách cache lại data sau khi lấy từ server về client
* Hiểu và áp dụng cách sử dụng các thư viện quản lí Global State trong ReactJS
* Hiểu và biết cách sử dụng GraphQL
* Tối ưu hoá các tính toán làm nặng bộ nhớ trong ứng dụng
* Biết được kiến trúc và các dịch vụ sử dụng của AWS trong Serverless App
  1. Hướng phát triển tiếp theo

Hiện tại, các tính năng em đã làm đều đang hoạt động ổn định, sắp tới trong phần code của phía UI em sẽ cần tối ưu thêm một vài thứ như: cấu trúc lại code, chia rõ mục đích các component, 1 vài hàm gọi API và xử lí dữ liệu đang được đặt trong component hiển thị giao diện. Điều này là không tốt vì sẽ gặp khó khăn trong việc quản lí, tái sử dụng và tìm lỗi. Cần tách biệt riêng đoạn code nào sẽ hiển thị giao diện và đoạn code nào sẽ xử lí data.

Trên đây là toàn bộ bài báo cáo của em.