**CHƯƠNG 1: GIỚI THIỆU CHUNG VỀ CÔNG TY TMA SOLUTIONS**

1. **Thông tin về đơn vị thực tập:**
   1. ***Quá trình hình thành và phát triển của công ty TMA Solutions:***

TMA Solutions được thành lập năm 1997, là công ty phần mềm hàng đầu Việt Nam. Với sự phát triển bền vững về nhân lực, hiện nay, TMA đã xây dựng được đội ngũ hơn 2400 kỹ sư trẻ, nhiệt huyết luôn nhận được sự tin cậy, đánh giá cao từ phía khách hàng. Mục tiêu của TMA là trở thành tập đoàn công nghệ cao hàng đầu và góp phần đưa tên Việt Nam vào bản đồ gia công phần mềm thế giới.

TMA Solutions hiện là đối tác tin cậy có nhiều năm kinh nghiệm thực hiện những dự án lớn & phức tạp với những công nghệ mới nhất với hơn 100 khách hàng đến từ hơn 27 quốc gia trên thế giới.



**Các trụ sở của TMA trên toàn thế giới:**

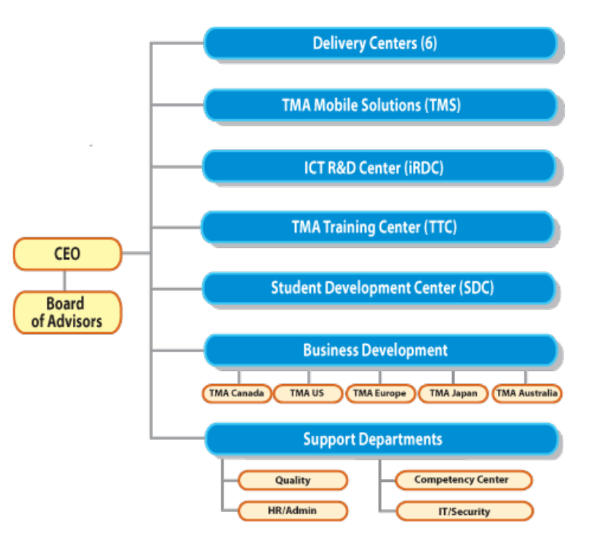
* Vietnam: Ho Chi Minh City
* USA: San Jose
* Canada: Ottawa
* Australia: Melbourne
* Japan: Tokyo

**Các giải thưởng, bằng khen đạt được:**

* Bằng khen của Ủy ban Nhân dân Thành phố Hồ Chí Minh: Có thành tích xuất sắc trong lĩnh vực CNTT-TT, góp phần tích cực vào sự phát triển CNTT-TT của Thành phố.
* Huy Chương Vàng Xuất khẩu Phần Mềm (15 năm liền từ 2004 đến 2018).
* Top 5 Đơn Vị Gia Công Xuất khẩu Phần Mềm Hàng Đầu năm 2009 đến 2018.
* Bằng khen của VINASA: Có thành tích xuất sắc, đóng góp cho các hoạt động của Hiệp hội và cho sự phát triển của ngành phần mềm, dịch vụ công nghệ thông tin Việt Nam.
* Là một trong 15 công ty hàng đầu thế giới trong việc áp dụng hiệu quả quy trình gia công phần mềm (Báo cáo của công ty tư vấn Mỹ Aberdeen, 09/2002).
* Đối tác chính thức của Microsoft từ 2007 đến 2018.
* Đạt nhiều chứng chỉ quốc tế cao cấp (CMMI-L5, TL 9000, ISO 9001:2000, ISO 27001:2013).
  1. ***Tổ chức và các lĩnh vực hoạt động của công ty:***

**TMA chuyên cung cấp các dịch vụ và giải pháp cho nhiều lĩnh vực:**

* Software Development
* Software Testing
* Porting & Migration
* Production Support
* IT Managed Services
* Visual Design
  1. ***Cơ cấu tổ chức quản lý sử dụng nguồn nhân lực:***

**Hình 1. Sơ đồ tổ chức TMA Solutions**

* 1. ***Tình hình hoạt động của doanh nghiệp trong 3-5 năm:***
     1. ***Tình hình cơ sở vật chất:***
     2. ***Tình hình sản xuất kinh doanh của công ty:***
     3. ***Phương hướng hoạt động trong thời gian tới:***

**Năm 2015:**

* Đạt 1800 nhân viên
* Ra mắt People CMM
* Ra mắt hệ thống truyền hình cho truyền thông nội bộ

**Năm 2016:**

* Ra mắt Techday 2016
* Thành lập dịch vụ phân tích kinh doanh(Business Analysis service )
* Ra mắt giải pháp thương mại điện tử, Microsoft và CRM

**Năm 2017:**

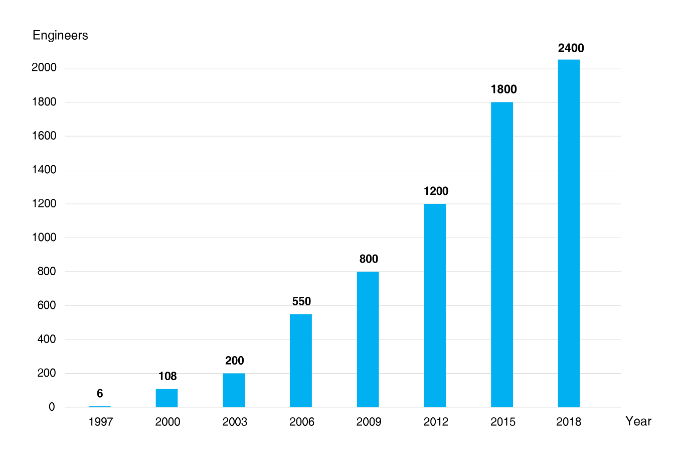
* Kỷ niệm 20 năm
* Đạt 2000 nhân viên
* Được trao huy chương vàng về gia công phần mềm trong 14 năm liên tiếp
* Thành lập TMA Innovation Center

**Năm 2018:**

* Thành lập trung tâm phần mềm tự đông TMA(TMA Automotive Software Center )
* Thành lập TMAsia in Singapore
* Thành lập TMA Binh Dinh
* Khởi công xây dựng TMA Innovation Park in Quy Nhon
* Thành lập TMA Blockchain Development Center
* Thành lập TMA DevOps Service Center

**Sự tăng trưởng về số lượng nhân viên:**

Sau hơn 21 năm tăng trưởng ổn định, quy mô đội ngũ kỹ thuật của công ty TMA Solutions đã tăng từ 6 lên 2.400 kỹ sư.

**Hình 2. Sự phát triển nhân lực bền vững từ 6 người đến hơn 2400 người (1997-2018)**

- Sơ lược về sự hình thành và phát triển của đơn vị.

- Tổ chức và các lĩnh vực hoạt động của đơn vị.

- Tổ chức quản lý sử dụng các nguồn lực của đơn vị.

- Cơ cấu tổ chức.

- Tình hình hoạt động kinh doanh của doanh nghiệp trong 3 - 5 năm.

- Các nội dung khác (tuỳ theo lĩnh vực của đề tài)

1. Thông tin về vị trí sinh viên tham gia thực tập:

- Giới thiệu chung về vị trí công tác.

- Đặc điểm, yêu cầu.

- Cơ cấu tổ chức, nhiệm vụ liên quan.

Nội dung bao gồm: Tóm tắt, hệ thống hoá một cách súc tích các thông tin có liên quan.

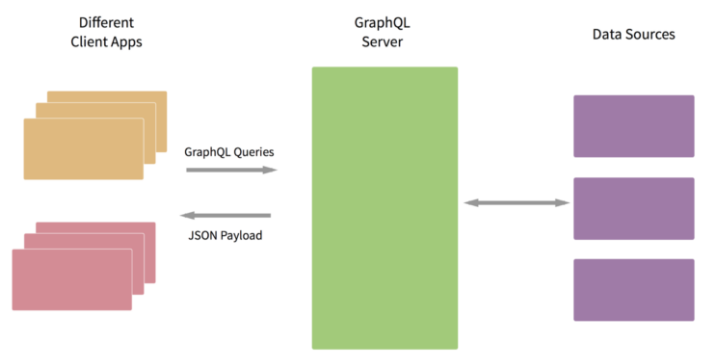
**CHƯƠNG 2: NỘI DUNG BÁO CÁO**

1. **Tìm hiểu về GraphQL và ứng dụng:**
   1. **Giới thiệu về GraphQL:**

GraphQL là một tiêu chuẩn API mới cung cấp một giải pháp thay thế hiệu quả, mạnh mẽ và linh hoạt hơn REST. Ban đầu GraphQL được Facebook phát triển và hiện là một open-source đang được một một cộng đồng lớn bao gồm các công ty và các cá nhân trên khắp thế giới chung tay xây dựng.

****

Như ta đã biết hiện tại API đã trở thành các thành phần phổ biến của cơ sở hạ tầng phần mềm. Một API định nghĩa cách client có thể tải dữ liệu từ server và GraphQL cho phép client có thể định nghĩa chính xác dữ liệu nào nó cần từ một API.

 Client truy vấn đến máy chủ GraphQL bằng các truy vấn với đặc điểm: format của dữ liệu trả về được mô tả trong câu truy vấn và được định nghĩa ở phía client thay vì ở server. Nói đơn giản hơn, đây là truy vấn hướng client, cấu trúc dữ liệu không khô cứng 1 khuôn mẫu từ server (REST API) mà thay đổi theo từng ngữ cảnh sao cho hiệu quả nhất đối với client mà chỉ cần dùng duy nhất 1 endpoint.

* 1. **Các thành phần trong GraphQL:**

GraphQL thực chất chỉ được phát hành dưới dạng specification, và có thể được implement bằng bất cứ ngôn ngữ lập trình nào, điều đó dẫn tới việc cần có một ngôn ngữ thống nhất chung mà mọi implementation của GraphQL đều phải tuân theo. Đối với GraphQL, ngôn ngữ chung này được sử dụng để viết schema - do đó nó được gọi là Schema Definition Langulage (SDL).

Để định nghĩa các chứng năng của app thông qua API, ta cần phải thêm một vài thứ nữa vào [root types](https://graphql.org/learn/schema/#the-query-and-mutation-types) của GraphQL Schema : Query, Mutation và Subscription. 3 loại type đặc biệt này định nghĩa đầu vào cho một GraphQL API.

Bất cứ một GraphQL server nào - dù được implement bằng ngôn ngữ gì - cũng đều phải tuân theo Schema nói trên, và client khi giao tiếp với server cũng sẽ chỉ cần hiểu được Schema là đủ - không cần quan tâm thực chất server được implement như thế nào.

GraphQL API được tạo ra từ 3 phần chính: schema, queries/mutations và resolvers ngoài ra còn có subscriptions.

### **Queries, mutations và subscription:**

### **Queries**

### Query là các câu lệnh lấy dữ liệu (tương tự method GET trong RestFul API)

### GraphQL sử dụng việc nạp dữ liệu khác với REST. Nó chí có duy nhất 1 single endpont và hoàn toàn phụ thuộc vào client để xác định những dữ liệu cần thiết. Vì thế client phải chỉ ra các trường cần thiết, như ví dụ dưới đây:

### 

### **Mutation**

### Các câu lệnh để thêm, xóa, sửa dữ liệu (tuơng tự method POST/PUT/DELETE trong RESTful API)

### Hầu hết các cuộc thảo luận về GraphQL tập trung vào tìm nạp dữ liệu, nhưng bất kỳ nền tảng dữ liệu hoàn chỉnh nào cũng cần một cách để sửa đổi dữ liệu phía máy chủ.

### Trong REST, bất kỳ yêu cầu nào cũng có thể gây ra một số tác dụng phụ trên máy chủ, nhưng theo quy ước, chúng tôi đề nghị một người không sử dụng các yêu cầu GET để sửa đổi dữ liệu. GraphQL tương tự - về mặt kỹ thuật, bất kỳ truy vấn nào cũng có thể được thực hiện để gây ra ghi dữ liệu. Tuy nhiên, thật hữu ích khi thiết lập một quy ước rằng bất kỳ hoạt động nào gây ra ghi nên được gửi rõ ràng thông qua một mutation.

### Giống như trong các truy vấn, nếu trường mutations trả về một loại đối tượng, bạn có thể yêu cầu các trường lồng nhau. Điều này có thể hữu ích để xem trạng thái mới của một đối tượng sau khi cập nhật. Hãy xem xét một mutaions ví dụ đơn giản:

### 

### **Subscriptions**

### Một yêu cầu quan trọng khác đối với nhiều ứng dụng hiện nay là phải có kết nối thời gian thực với máy chủ để được thông báo ngay lập tức về các sự kiện quan trọng. Đối với trường hợp sử dụng này, GraphQL cung cấp khái niệm subscriptions.

### Khi một client subscriptions vào một sự kiện, nó sẽ bắt đầu và giữ một kết nối ổn định đến máy chủ. Bất cứ khi nào sự kiện cụ thể đó thực sự xảy ra, máy chủ sẽ đẩy dữ liệu tương ứng đến máy khách. Không giống như các queries và mutations tuân theo một chu kỳ “request-response-cycle”, các subscriptions đại diện cho một luồng dữ liệu được gửi đến máy khách.

### Subscriptions được viết bằng cách sử dụng cú pháp tương tự như queries và mutations. Dưới đây là một ví dụ nơi chúng tôi đăng ký các sự kiện xảy ra trên Person type:

### 

### Sau khi khách hàng gửi subscription này đến máy chủ, một kết nối được mở giữa họ. Sau đó, bất cứ khi nào một đột biến mới được thực hiện tạo ra một Person, máy chủ sẽ gửi thông tin về người này cho khách hàng:

### 

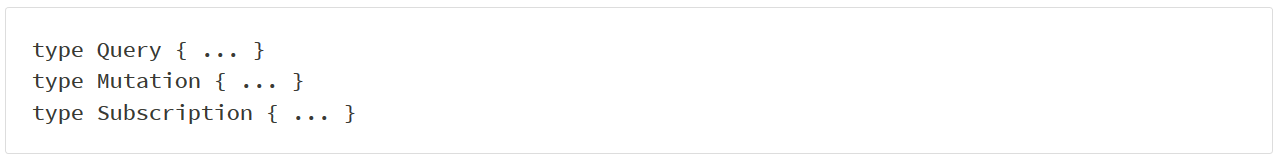
* + 1. Schema:

GraphQL sử dụng một hệ thống strong type để định nghĩa khả năng của một API. Tất cả các kiểu dữ liệu trong một API được định nghĩa trong một schema bằng SDL (Schema Definition Language) của GraphQL. Schema là một trong những khái niệm quan trọng nhất khi làm việc với API GraphQL. Schema này đóng vai trò là quy ước giữa client và server xác định cách mà client có thể truy cập dữ liệu.

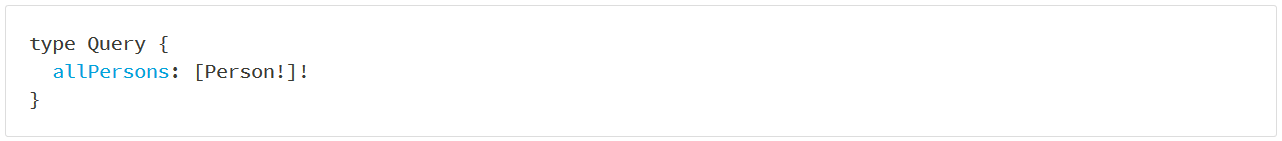
Sau khi schema được định nghĩa, các nhóm làm việc trên frontend và backend có thể thực hiện công việc của mình mà không cần liên lạc thêm vì cả hai đều biết cấu trúc xác định của dữ liệu được gửi qua mạng.

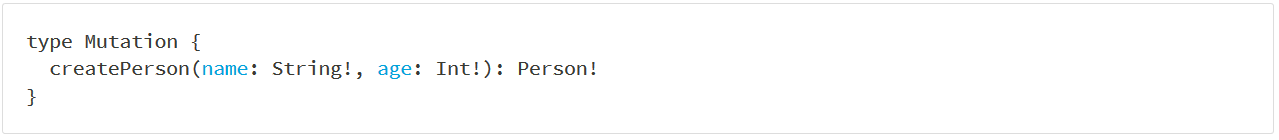
Các nhóm Frontend có thể dễ dàng kiểm tra các ứng dụng của họ bằng cách giả lập các cấu trúc dữ liệu cần thiết được trả về. Khi server đã sẵn sàng, các ứng dụng client có thể ngay lập tức chuyển sang sử dụng API thực tế được cung cấp bởi server.

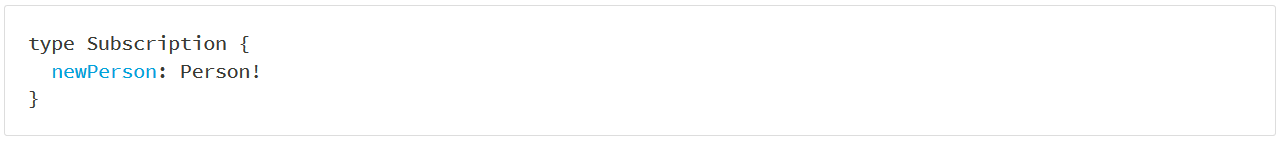
Nói chung, một schema chỉ đơn giản là một tập hợp các type của GraphQL. Tuy nhiên, khi viết schema cho API, có một số root type đặc biệt:



Các Query, Mutation, and Subscription type là các điểm truy cập cho các yêu cầu được gửi bởi client. Các type trên sẽ được viết như sau:



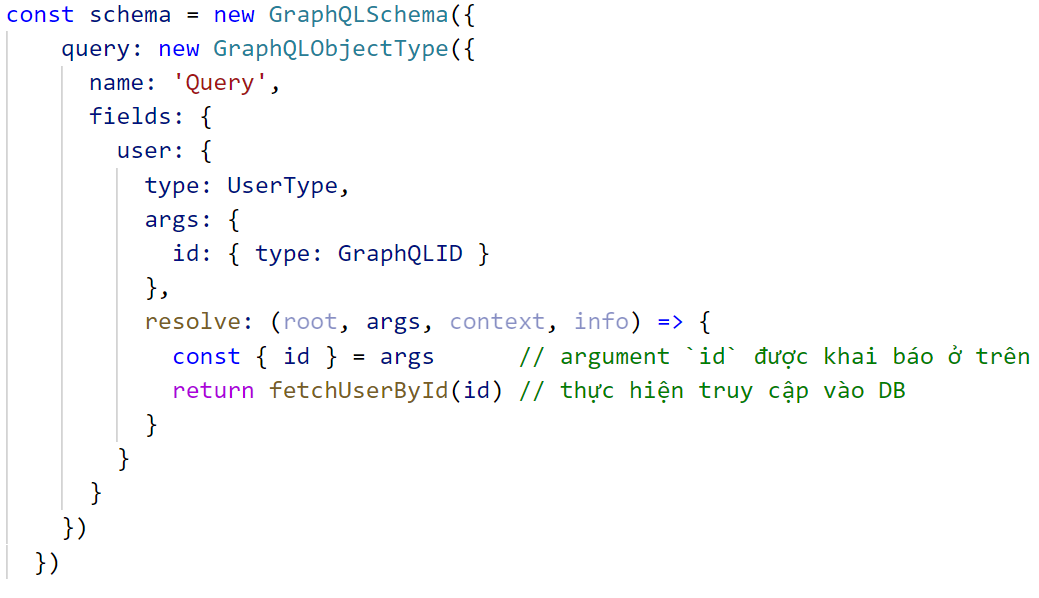




* + 1. Resolvers:

Ở dạng đơn giản nhất, một GraphQL server sẽ có một hàm resolver cho mỗi một field trong schema của nó. Mỗi resolver biết cách lấy dữ liệu cho trường tương ứng của nó. Khi mà một query của GraphQL có bản chất chỉ là một tập hợp các trường, việc mà một GraphQL Server cần làm khi xử lý một request đó là gọi và thực hiện tất cả các resolver tương ứng với các trường trong query.

Ta cùng xem ví dụ sau:



Định nghĩa 4 arguments được truyền vào resolver:

* **root**: Nhớ lại rằng ở trên ta đã nói rằng một GraphQL Server thực hiện một query bằng cách gọi tới các resolver tương ứng với các trường. GraphQL thực hiện điều này theo thuật toán [BFS](https://en.wikipedia.org/wiki/Breadth-first_search), đồng thời argument root ở mỗi lần gọi resolver chính là kết quả của lời gọi ở nút phía cha (do đó với nút đầu tiên thì root có giá trị là null).
* **args**: argument này chứa các parameter dùng cho query, trong trường hợp này là id của User cần lấy về.
* **context**: một object được truyền xuyên suốt chuỗi resolver mà mỗi resolver đều có thể ghi vào và đọc thông tin ra (nói chung là 1 nơi để các resolver giao tiếp và chia sẻ thông tin)
* **info**: chứa AST của chính câu query hay mutation.
  1. **Tính năng của GraphQL:**

#### **Không còn thừa hay thiếu dữ liệu**

Một trong những vấn đề phổ biến nhất với REST là sự dư thừa cũng như thiếu hụt dữ liệu. Điều này xảy ra vì cách duy nhất để client lấy dữ liệu là gọi đến endpoint và server trả về dữ liệu với cấu trúc cố định. Rất khó để thiết kế API theo cách có thể cung cấp cho client những dữ liệu chính xác với nhu cầu của tất cả các client.

* Sự dư thừa xảy ra khi một client phải tải xuống nhiều thông tin hơn so với nhu cầu thực sự của nó.
* Sự thiếu hụt dữ liệu xảy ra khi server không cung cấp đủ thông tin cần thiết trong một endpoint. Client sẽ phải gọi đến các endpoint bổ sung để lấy những dữ liệu còn thiếu. Điều này có thể dẫn đến tình huống mà client phải tải xuống một danh sách các phần tử, và tiếp đó phải gọi thêm endpoint cho mỗi phần tử để lấy thêm những dữ liệu cần thiết.

Với GraphQL, client chỉ cần gửi một câu truy vấn tới GraphQL server xác định rõ những dữ liệu cần thiết. Và server sau đó sẽ trả về cho client một đối tượng JSON chỉ chứa những dữ liệu mà client đã yêu cầu.

#### **Phát triển cập nhật frontend nhanh chóng**

Một pattern chung cho REST API là việc thiết kế cấu trúc các endpoint theo các view bên trong ứng dụng của mình. Điều này rất tiện lợi vì nó cho phép client nhận tất cả các thông tin cần thiết cho một view cụ thể bằng cách truy cập vào endpoint tương ứng.

Tuy nhiên hạn chế của cấu trúc này là nó không cho phép một sự thay đổi nhanh trên frontend. Mọi thay đổi về frontend đều có khả năng cao dẫn đến sự thay đổi lượng dữ liệu cần thiết (nhiều hơn hoặc ít hơn). Do đó, phía backend cũng cần được điều chỉnh để đáp ứng các nhu cầu dữ liệu mới. Điều này làm giảm năng suất và làm chậm sự kết hợp phản hồi của người dùng vào sản phẩm.

Với GraphQL, vấn đề này được giải quyết. Nhờ tính chất linh hoạt của GraphQL, client có thể chỉ định chính xác và cụ thể các yêu cầu về dữ liệu, phía backend không cần phải thay đổi gì khi phía frontend thay đổi về thiết kế và dữ liệu cần lấy về.

#### **Khả năng phân tích nhu cầu về dữ liệu**

Vì với GraphQL, mỗi client chỉ định chính xác thông tin mà nó cần nên ta có thể nắm được thông tin chi tiết về cách mà dữ liệu sẵn có đang được sử dụng. Dựa vào những thông tin này, ta có thể cải tiến các API, ví dụ bỏ đi các trường dữ liệu không được bất kỳ client nào yêu cầu.

Thêm vào đó, ta cũng có thể thực hiện giám sát hiệu năng xử lý request từ client của server. GraphQL sử dụng khái niệm về *resolver function* để thu thập dữ liệu được yêu cầu từ client. Việc đo lường và đánh giá hiệu suất của các *resolver* này cung cấp thông tin chi tiết quan trọng về các bottleneck trong hệ thống.

1. **Tổng quan về quá trình thực tập:**

Có bao nhiêu tuần thực tập thì thực hiện mỗi tuần như sau:

Nội dung thực tập mỗi ngày được giao bao nhiêu task (mỗi task [kèm theo mã công việc] được ước lượng bao nhiêu giờ phải hoàn thành, mô tả task, chụp hình giao diện hoặc hệ thống quản lý task).

Số lượng task hoàn thành (hoàn thành theo nghĩa người nghiệm thu cho pass)

Số lượng task không hoàn thành, lý do

Số người làm chung, vai trò

**CHƯƠNG 3: NHẬN XÉT, ĐÁNH GIÁ THỰC TRẠNG**

Nội dung bao gồm

3.1. Các nhận xét, đánh giá thực trạng của quá trình làm việc.

3.2. Các kiến nghị (nếu có)

**\* KẾT LUẬN**