```
app.get("/product",(req, res) => {res.json (products);
});
app.get("/product/:id", (req, res) => {
var newProduct = products.filter(function(product) {
if (product.id==req.params.id) {return true}
});
if (newProduct.length == 1) {
res.json(newProduct[0])
} else {
res.status(404);
res.json({ message: "Not Found" });}
});
```

```
app.post("/product", (req, res) => {
var newProductID = products[products.length - 1].id + 1;
let newProduct = {
        id: newProductID,
        productName: req.body.productName,
        quality: req.body.quality,
        price: req.body.price,   };
      products.push(newProduct);
      res. json ({
         message: "Product successfully added!",
         product : newProduct  });
});
```

```
app.get("/search/:songay", (req, res) => {
const songay = req.params.songay
if(songay <= 1) {
res.json("So ngay ko hop le")
} else{
var newTours = Tours.filter(function(tour) {
if (tour.songay==songay) {return true}
});
if (newTours.length >= 1) {
res.json(newTours)
} else {
res.status(404).json({ message: "Not Found" });}
});
}
```

```
app.put("/product/:id", (req, res) => {
  let oldProduct = products.find(item => item.id == req.params.id);
  if (!oldProduct) {
    res. json ({message: "Product not found!" });  }
  else{
    let index = products.indexOf(oldProduct);
    let product = products[index]
    product = {
      id: req.params.id,
      productName: req.body.productName,
      quality: req.body.quality,
      price: req.body.price,  };
    res. json ({ message: "product updated!",
      product })  }
});
```

```
app.delete("/product/:id", (req, res) => {
  let indexProduct = products.find(item => item.id == req.params.id);
  if (!indexProduct) {
    res. json ({message: "Product not found!"
    }); }
  else{
    let index = products.indexOf(indexProduct);
    let product = products[index]
    products.splice(indexProduct, 1);
    res. json ({
      message: "Product successfully deleted!",
      product})  }
});
```

Có rất nhiều cách để thực hiện tìm kiếm:

- Thông qua công cụ tìm kiếm back-end như Elasticsearch hoặc Solr.
- Các dịch vụ tìm kiếm thương mại như Algolia và AWS Cloudsearch.
- Sử dụng cơ sở dữ liệu với tìm kiếm tích hợp sẵn như MySQL hoặc MongoDB.
- Nền tảng ứng dụng và dịch vụ cơ sở dữ liệu với chức năng tìm kiếm bổ trợ, chẳng hạn như Firebase và Cloudant.
- Các thư viện tìm kiếm JavaScript phía máy khách như FlexSearch và Elasticlunr.
- Thư viện JavaScript phía máy khách đồng bộ hóa dữ liệu với cơ sở dữ liệu phụ trợ: Ví dụ: PouchDB.
- Google Custom Search Engine.(Công cụ Tìm kiếm Tuỳ chỉnh của Google.)

Chỉ mục tìm kiếm là gì? Ở dạng đơn giản nhất, trình lập chỉ mục tìm kiếm giải quyết vấn đề này bằng cách phân tích tập dữ liệu và xây dựng chỉ mục các cụm từ tìm kiếm (từ hoặc cụm từ) và vị trí của chúng trong dữ liệu - giống như một chỉ mục ở cuối sách. Sau đó, triển khai tìm kiếm có thể tra cứu truy vấn trong chỉ mục thay vì quét tất cả dữ liệu. Người lập chỉ mục cũng có thể triển khai các tính năng như xử lý dừng từ và xuất phát từ gốc.

Chỉ mục (Index) là các cấu trúc dữ liệu đặc biệt, lưu giữ một phần nhỏ của tập hợp dữ liệu, giúp việc truy vấn dữ liệu vào Collection một cách dễ dàng hơn. Chỉ mục lưu giữ giá trị của một trường cụ thể hoặc tập hợp các trường, được sắp xếp bởi giá trị của trường như đã được xác định trong chỉ mục.

Tạo một bảng lưu các thông tin của Các chữ trong database (được lập chỉ mục) cùng với các bản ghi mà nó được tham chiếu đến , Các chữ được đánh số index , Khi người dung sreach chữ đó , thì hệ thống sẽ tách các chữ và lấy ra những index đã được đánh dấu. Từ các index đó truy cập đến các bảng ghi đã được đánh đấu tương ứng với index . Từ đó tìm kiếm với các bảng ghi được lấy ra

Dùng search engine Elastic Search (ES)

- Xây dựng theo cơ chế RESTful API, hoạt động như 1 Web Server độc lập
- Có tốc độ tìm kiếm lớn do sử dụng kỹ thuật Inverted Index
- Cho phép bạn thực hiện các câu truy vấn từ đơn giản tới phức tạp, từ có cấu trúc tới phi cấu trúc theo bất kì cách bạn muốn
- Cho khả năng thống kê, phân tích dữ liệu dễ dàng
- Được thiết kế phân tán, bao gồmđảm bảo tốc độ và tin cậy (hệ thống vẫn hoạt động khi một node bị sập)

Cơ chế lưu trữ của elasticsearch là lưu trữ dưới dang các term có cấu trúc ison. hiểu đơn giản là phân tích và chia vùng dữ liệu. từ khóa nào nằm ở khu vực nào. những từ cùng nghĩa hoặc tương tự sẽ cùng ở một vùng.

KHÁI NIỆM : MVC là viết tắt của cụm từ **"Model-View-Controller"**. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. MVC là một mẫu kiến trúc phần mềm để tạo lập giao diện người dùng trên máy tính. MVC chia thành ba phần được kết nối với nhau như tên gọi: **Model (dữ liệu), View (giao diện)** và **Controller (bộ điều khiển).**

ƯU ĐIỂM :Đầu tiên, nhắc tới ưu điểm mô hình MVC thì đó là băng thông (Bandwidth) nhẹ vì không sử dụng viewstate nên khá tiết kiệm băng thông. Việc giảm băng thông giúp website hoạt động ổn định hơn.

Kiểm tra đơn giản và dễ dàng, kiểm tra lỗi phần mềm trước khi bàn giao lại cho người dùng.

Một lợi thế chính của MVC là nó tách biệt các phần Model, Controller và View với nhau.

Sử dụng mô hình MVC chức năng Controller có vai trò quan trọng và tối ưu trên các nền tảng ngôn ngữ khác nhau

Ta có thể dễ dàng duy trì ứng dụng vì chúng được tách biệt với nhau.

Có thể chia nhiều developer làm việc cùng một lúc. Công việc của các developer sẽ không ảnh hưởng đến nhau.

Hỗ trợ **TTD (test-driven development).** Chúng ta có thể tạo một ứng dụng với unit test và viết các won test case.

Phiên bản mới nhất của MVC hỗ trợ trợ thiết kế responsive website mặc định và các mẫu cho mobile. Chúng ta có thể tạo công cụ View của riêng mình với cú pháp đơn giản hơn nhiều so với công cụ truyền thống.

NHƯỢC ĐIỂM : Bên cạnh những ưu điểm MVC mang lại thì nó cũng có một số nhược điểm cần khắc phục.

MVC đa phần phù hợp với công ty chuyên về website hoặc các dự án lớn thì mô hình này phù hợp hơn so với với các dự án nhỏ, lẻ vì khá là cồng kềnh và mất thời gian.

Không thể **Preview** các trang như ASP.NET.

Khó triển khai.

Ưu điểm N-tier: -Server có thể phân tán tự do

-Bảo mật: Bạn có thể bảo mật từng lớp trong ba lớp riêng biệt bằng các phương pháp khác nhau.

-Dễ quản lý: Bạn có thể quản lý từng cấp riêng biệt, thêm hoặc sửa đổi từng cấp mà không ảnh hưởng đến các cấp khác.

-Có thể mở rộng: Nếu bạn cần thêm nhiều tài nguyên hơn, bạn có thể thực hiện theo từng cấp mà không ảnh hưởng đến các cấp khác.

-Linh hoạt: Ngoài khả năng mở rộng riêng biệt, bạn cũng có thể mở rộng từng cấp theo bất kỳ cách nào mà yêu cầu của bạn quy định.

-Phát triển hiệu quả hơn. Kiến trúc N-tier rất thân thiện để phát triển, vì các nhóm khác nhau có thể làm việc trên mỗi tầng. Bằng cách này, bạn có thể chắc chắn rằng các chuyên gia thiết kế và trình bày làm việc trên tầng trình bày và các chuyên gia cơ sở dữ liệu làm việc trên tầng dữ liệu.

-Dễ dàng thêm các tính năng mới. Nếu bạn muốn giới thiệu một tính năng mới, bạn có thể thêm nó vào tầng thích hợp mà không ảnh hưởng đến các tầng khác.

Luồng xử lý trong của mô hình **MVC,** bạn có thể hình dung cụ thể và chi tiết qua từng bước dưới đây:

Khi một yêu cầu của từ máy khách (Client) gửi đến Server. Thì bị Controller trong MVC chặn lại để xem đó là URL request hay sự kiện.

Sau đó, **Controller** xử lý **input** của user rồi giao tiếp với **Model** trong MVC.

Model chuẩn bị data và gửi lại cho Controller.

Cuối cùng, khi xử lý xong yêu cầu thì Controller gửi dữ liệu trở lại View và hiển thị cho người dùng trên trình duyệt.

Ở đây**, View không giao tiếp trực tiếp với Model.** Sự tương tác giữa **View** và **Model** sẽ chỉ được xử lý bởi **Controller**.

KHÁI NIỆM : Kiến trúc N-tier còn được gọi là multi-tier architecture, một phương pháp kiến trúc ứng dụng trong phát triển phần mềm. Nó thích hợp cho việc xây dựng các ứng dụng lớn, đặc biệt là các ứng dụng doanh nghiệp, các ứng dụng đòi hỏi tính **scalability, security, fault tolerance, reusability** và **maintainability**.

Gọi là N-tier điều đó có nghĩa kiến trúc này có thể có 1, 2, 3 hoặc hơn số các layer phụ thuộc vào cách phân chia kiến trúc hệ thống.

3-Tier vẫn là mẫu phổ biến nhất và được định nghĩa cụ thể về trách nhiệm từng tier như sau:

Tầng Presentation: Chính là Giao diện người dùng, đây chính là phần mềm ứng dụng mà người dùng sẽ thấy và tương tác (Có thể là Website hoặc Mobile App, hoặc Window app). Khi người dùng nhập thông tin họ cần. Hành động người dùng được xử lí đi qua các tầng Logic, tầng Data.

Tầng Logic: Đây là tầng chứa tất cả những phương pháp xử lý, đọc & ghi dữ liệu trước khi đưa đến UI người dùng, nó những gì được cho phép trong ứng dụng của bạn.

Tầng Data: Tầng Data là nơi lưu trữ tất cả dữ liệu trong ứng dụng, tại tầng này bạn thực hiện các phương thức lưu trữ dữ liệu vào DB, triển khai các giải pháp bảo mật, transaction cần thiết.

Ưu điểm N-tier: -Dễ dàng tái sử dụng. Vì ứng dụng được chia thành các tầng độc lập nên bạn có thể dễ dàng sử dụng lại từng tầng cho các dự án phần mềm khác. Ví dụ: nếu bạn muốn sử dụng cùng một chương trình, nhưng cho một tập dữ liệu khác, bạn chỉ có thể sao chép các tầng logic và trình bày, sau đó tạo một tầng dữ liệu mới.

Khó khăn N-tier:

-Độ hiệu quả phụ thuộc vào mạng và hệ thống nên khó lường trước.

-Nếu các server được quản lý bởi các tổ chức khác nhau thì có vấn đề về quản lý chúng (Nếu 1 trong hai là mongoDB hoặc cloudinary bị lỗi, thì hệ thống không hoạt động).

-Phụ thuộc vào nhà cung cấp dịch vụ

-Yêu cầu đảm bảo băng thông nhanh, yêu cầu phần cứng xử lý tốt, bảo trì thường xuyên

-Tốn nhiều chi phí (triển khai và bảo trì)

-Thời gian xử lý có thể bị chậm trễ

3-Tier trông có vẻ khá giống MVC nhưng sự thật lại không phải vậy. Nguyên tắc cơ bản của 3-Tier Architecture đó là tầng Presentation/UI sẽ không bao giờ giao tiếp trực tiếp tới tầng cuối (Data Tier), mà buộc phải đi qua các tầng giữa (Middle Tier). Trong khi đó, các thành phần của MVC lại giao tiếp với nhau theo mô hình **Tam giác:** việc User tương tác với View sẽ đi tới Controller, Controller sẽ xử lí cập nhật Model và đẩy Model vào View, khi này View được cập nhật trực tiếp từ Model.

Giao tiếp giữa các tầng trong 3-Tier Architecture: Trong mô hình 3-Tier, mỗi lớp sẽ nằm tách biệt, như tầng Presentation/UI sẽ nằm trên một vài Server, chúng giao tiếp với back-end App Server chịu trách nhiệm thực hiện các Business Logic và nói chuyện với DB Server, (một vài App Server còn tương tác thêm với các dịch vụ từ xa khác. Ví dụ như Authorize để xử lý payment) . Đôi khi trong một số hệ thống, người ta thêm vào các tầng khác vì vậy mới sinh ra khái niệm N-Tier.

N-Tier và MVC có thể được sử dụng cùng trong một ứng dụng, đã có rất nhiều ứng dụng áp dụng điều này. Ví dụ, bạn có thể sử dụng kiến trúc N-tier là kiến trúc tổng thể, sử dụng MVC trong tầng Presentation.

***2-Tier Architecture**:* cũng như kiến trúc Client – Server, các giao tiếp chỉ xảy ra giữa Client – Server. Nó bao gồm Presentation layer (UI) chạy phía client và 1 tầng Data layer làm nhiệm vụ thực thi, lưu trữ phía Server.

Không có tầng Business Logic hay tầng trung gian nào giữa Client và Server. Đây có lẽ là kiến trúc mà đa số các bạn "vô tình" sử dụng mà không biết khi còn là Newbie chưa có khái niệm nhiều về Architecture. Đơn giả như chúng ta xây dựng các ứng dụng web nhỏ với 1 đống code connection, query trực tiếp và DB lấy dữ liệu đổ ra view ..v.v.v..

***1-Tier Architecture**:* Đây là loại đơn giản nhất, tiêu biểu là các ứng dụng chạy trên máy tính cá nhân của bạn. Tất cả các phần tử của ứng dụng chỉ chạy trên máy hoặc server đơn lẻ.