****

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP HỒ CHÍ MINH**

----------------------

**Khoa: Công Nghệ Thông Tin**



**BÁO CÁO BÀI TẬP NHÓM**

***Đề tài:* XÂY DỰNG WEBSITE QUẢN LÝ BẤT ĐỘNG SẢN**

Mã HP : 420300212602

Môn học : Công nghệ và lập trình tích hợp nâng cao

Ngày báo cáo : 11/11/2021

Giáo viên HD : Trương Bá Phúc

Mục Lục

[I. Thông tin chung về đề tài 3](#_Toc87904130)

[**1.** **Tên đề tài:** 3](#_Toc87904131)

[**2.** **Danh sách sinh viên tham gia thực hiện:** 3](#_Toc87904132)

[**3.** **Tìm hiểu đề tài** 3](#_Toc87904133)

[**4.** **Mục tiêu đề tài** 4](#_Toc87904134)

[II. Nền tảng kiến thức 4](#_Toc87904135)

[**A.** **Views** 4](#_Toc87904136)

[**1.** **EJS Template Engine** 4](#_Toc87904137)

[2. **Cách dùng EJS** 5](#_Toc87904138)

[3. **Truyền giá trị từ route sang file .ejs** 8](#_Toc87904139)

[**B.** **Model** 10](#_Toc87904140)

[**1.** **NodeJS** 10](#_Toc87904141)

[**2.** **ExpressJS** 10](#_Toc87904142)

[**3.** **HTTP request methods** 12](#_Toc87904143)

[**4.** **HTML Form Encoding** 13](#_Toc87904144)

[**5.** **RESTful API** 13](#_Toc87904145)

[**C.** **DATABASE** 14](#_Toc87904146)

[**1.** **Big data** 14](#_Toc87904147)

[**2.** **NoSQL** 15](#_Toc87904148)

[**3.** **MongoDB** 16](#_Toc87904149)

[**4.** **Mongoose** 17](#_Toc87904150)

[**III.** **Kết quả đạt được** 19](#_Toc87904151)

[**1)** **Mô hình Cơ Sở Dữ Liệu:** 19](#_Toc87904152)

[**2)** **Kết nối Database** 19](#_Toc87904153)

[**3)** **Giao diện chính của Website** 20](#_Toc87904154)

[**4)** **Chức năng CRUD Building** 23](#_Toc87904155)

[**5)** **Chức năng CRUD District** 29](#_Toc87904156)

[**6)** **Chức năng CRUD Rent Type** 31](#_Toc87904157)

[**IV.** **Nhận xét và cải tiến** 34](#_Toc87904158)

# Thông tin chung về đề tài

1. **Tên đề tài:**

Xây dựng Website quản lý bất động sản.

1. **Danh sách sinh viên tham gia thực hiện:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **MSSV** | **HỌ TÊN** | **LỚP** | **PHÂN CÔNG** |
| 1 | 17068721 | Nguyễn Trung Thành |  | * Công việc chính: Xây dựng phần backend cho Website |
| 2 | 17066391 | Đỗ Minh Tuấn |  | * Công việc chính: Xây dựng database và logic hệ thống, viết báo cáo kết hợp với hỗ trợ bạn xây dựng phần backend |
| 3 | 16070491 | Đỗ Văn Đỉnh | DHCNTT12C | * Công việc chính: Xây dựng phần frontend cho Website |

1. **Tìm hiểu đề tài**

Thị trường bất động sản Việt Nam đang trên đà phát triển và giữ tỷ trọng ngày càng lớn trong cơ cấu nền kinh tế. Tuy nhiên, thị trường này lại đang phát triển không đồng bộ, chưa hoàn thiện về cơ cấu tổ chức và vận hành thị trường, dẫn đến tình trạng thiếu minh bạch về thông tin và yếu tố đầu cơ còn lớn. chính lúc này việc phát triển hoạt động định giá không chỉ là một đòi hỏi tất yếu mà còn rất cấp bách đối với Việt Nam.

Hoạt động kinh doanh bất động sản mà đặc biệt là mảng kinh doanh dịch vụ bất động sản ra đời với những hình thức kinh doanh ngày càng đa dạng và phong phú đã góp phần không nhỏ trong việc thúc đẩy thị trường bất động sản Việt Nam phát triển lành mạnh, đúng hướng. Nắm bắt được xu hướng phát triển nóng bỏng của nghề định giá bất động sản trong tương lai gần, Công ty cổ phần sàn giao dịch bất động sản Việt Nam được thành lập với chức năng chuyên về kinh doanh dịch vụ về bất động sản. Trong đó, dịch vụ định giá bất động sản là dịch vụ nòng cốt hỗ trợ tất cả các dịch vụ khác như tư vấn, môi giới, đấu giá, quảng cáo bất động sản phát triển.

Nhờ ban lãnh đạo và đội ngũ cán bộ lành nghề và tâm huyết, luôn cố gắng hết mình với mục tiêu chung là phát triển công ty thành một trong những công ty đi đầu trong lĩnh vực định giá, trở thành sàn giao dịch bất động sản số 1 miền Bắc, là địa chỉ tin cậy cho các dự án bất động sản và giới kinh doanh bất động sản cũng như những người có nhu cầu mua, bán bất động sản. Tuy mới thành lập nhưng Công ty cổ phần sàn giao dịch bất động sản Việt Nam đã có nhiều hợp đồng giá trị lớn, thực hiện nhiều giao dịch bất động sản. Tuy nhiên, công tác định giá tại công ty chưa được quan tâm, còn một số vướng mắc và hạn chế.

Xuất phát từ chính những vấn đề lý luận và thực tiễn nêu trên em xin lựa chọn đề tài: “Website xây dựng quản lý bất động sản” để nghiên cứu làm chuyên đề cho bài tập lớn.

1. **Mục tiêu đề tài**

- Ở đây website bất động sản này cho người dùng có thể truy cập website để xem nhà, đất để thuê, hoặc có thể là mua.

- Hệ thống website gồm có 3 thành phần chính:Building, District, Rent Type

- Building đóng vai trò là một đối tượng trọng tâm của website.

# Nền tảng kiến thức

1. **Views**
2. **EJS Template Engine**

[**EJS**](https://ejs.co/)viết tắt của Embedded JavaScript – nhúng javascript hay javascript được nhúng nhỉ? Thực chất chỉ là javascript (JS) thôi, như những gì chúng ta đã học về JS, nhưng mà ở đây, ejs sử dụng JS cơ bản để nhúng vào trong code HTML nhằm mục đích tạo mã đánh dấu (markup) – từ đó tạo ra file .HTML khi chạy.

* Khi tạo ứng dụng express với epxress-generator sử dụng view engine ejs thì ejs **đã được cài và config**trong ứng dụng của chúng ta rồi, các file .ejs này nằm trong thư mục views với format: **tên\_file.ejs**.
* Nếu chỉ sử dụng express thuần bằng cách gọi module này vào sử dụng thì ta cài đặt ejs bằng cú pháp:

npm install ejs

, sau đấy trong file index.js thêm 2 dòng config để set ejs cho ứng dụng:

app.set**(**'views', path.join**(**\_\_dirname, 'views'**))**;

app.set**(**'view engine', 'ejs'**)**;

const https = require**(**'https'**)**;

const express = require**(**'express'**)**;

const path = require**(**'path'**)**;

const app = express**()**;

const port = 3000;

app.set**(**'views', path.join**(**\_\_dirname, 'views'**))**;

app.set**(**'view engine', 'ejs'**)**;

app.get**(**'/user', **function** **(**req, res**)** **{**

res.render**(**'index', **{**data: 'John Does'**})**;

**})**

Ta tạo thư mục “**views**” trong ứng dụng là thư mục sẽ chứa các file ejs.

Đối với cả 2 cách trên, để hiển thị ra HTML, ta sẽ render file .ejs và truyền giá trị (nếu có) cho nó (tên file không cần .ejs):

res.render**(**'path/file\_name', **{**variable: value**})**;

1. **Cách dùng EJS**

* **Cách nhúng code JS**

Chúng ta nhúng JS vào trong file ejs bằng cú pháp: **<% <js\_code> %>**

Ví dụ:

**<div>**

**<div>**

**<p>**

<% var myName = "John Does"%>

Hello <%-myName%>!

**</p>**

**</div>**

**</div>**

Như các bạn thấy ở ví dụ trên, đoạn code JS gán “John Does” cho biến **myName**được đặt trong **<% %>**, lúc này ejs sẽ hiểu và lưu vào bộ nhớ, sau đấy khi xuất ra kết quả, ta chỉ cần gọi biến đấy ra <%-myName%>

Graphical user interface, text, application

Description automatically generated

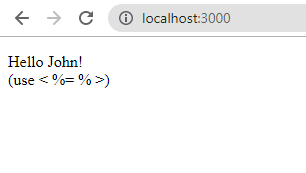
* **Xuất giá trị**

– Xuất giá trị “**unescaped**” (không lưu vào bộ nhớ đệm)

Lấy lại ví dụ trên, sau khi gán giá trị cho biến myName, ta thực hiện cú pháp: **<%- <value>%>**, ở đây ta có<%-myName%, nó sẽ xuất ra giá trị của biến **myName**.

– Xuất giá trị “**escaped**” (lưu vào bộ nhớ đệm)

Thay vì sử dụng cú pháp <%- <value>%>, ta sẽ sử dụng: **<%= <value>%>**



Ta thấy kết quả hoàn toàn giống nhau. Hãy thử chạy đoạn code sau:

**<div>**

**<div>**

**<p>**

<% var myName = "D &amp; G"%>

Hello <%-myName%>! (&lt;%-%&gt;)

**<br>**

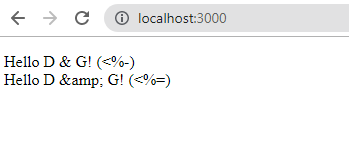
Hello <%=myName%>! (&lt;%=%&gt;)

**</p>**

**</div>**

**</div>**

Chạy thử:



Kết quả khác nhau. Trong khi, **<%-** cho ra giá trị của “&” còn <%= cho ra giá trị ban đầu (&amp;).

Bởi vì <%= sẽ không hiển thị mã HTML đó mà hiển thị nó dưới dạng string để tránh các cuộc tấn công XSS.

Với <% -, bạn có thể tránh điều này và nó sẽ hiển thị mã HTML.

* **Ghi chú, comment code**

Bình thường trong JS, chúng ta sử dụng “//” để code được bỏ qua khi chạy. Trong ejs ta sử dụng: <%#comment%>

* **Sử dụng lại HTML**

Ví dụ bạn có 1 ứng dụng nhỏ tầm 5 trang, mỗi trang đều có header và footer giống nhau, như vậy, 5 trang thì bạn phải copy + paste cho 4 trang còn lại, mỗi lần chỉnh sửa header hay footer thì phải copy + paste lại – khá là mất thời gian. Với cú pháp **<% include(‘path/file’) %>** sẽ giúp chúng ta tiết kiệm thời gian, nhanh, gọn để sử dụng lại code html từ các file khác, nếu có chỉnh sửa thì chỉ cần update ở file gốc là được.

Ví dụ:

**<**%- include**(**'../path/header'**)**-%**>**

Hello!

**<**%- include**(**'../path/footer'**)**-%**>**

Ở trên, ta đã gọi 2 file header và footer vào trong 1 file khác. Rất là tiện lợi.

* **Sử dụng JS cơ bản**

Trong ejs, ta có thể sử dụng vòng lặp, câu điều kiện,… một cách rất linh hoạt. Ví dụ:

**<**%

var arrN = **[**1,2,6,8,7**]**;

arrN.forEach**(**item =**>** **{**

**if** **(**item === 1**)** **{**

%**>**

**<**h1**>This** is h1 tag**<**/h1**>**

**<**%**}** **else** **if** **(**item === 2**){**%**>**

**<**h2**>This** is h2 tag**<**/h2**>**

**<**%**}** **else** **{**%**>**

**<**p**>**Not h tag**<**/p**>**

**<**%**}**%**>**

**<**%**})**%**>**

Chỉ cần nằm trong <% %>, code JS sẽ được thực thi.

1. **Truyền giá trị từ route sang file .ejs**

Khi xử lý backend, mỗi request của client sẽ khớp với một route mà chúng ta đã định nghĩa và thực hiện các function xử lý. Ta sẽ phải truyền giá trị từ các function này sang ejs để nhúng vào code HTML để hiển thị một cách linh động (dynamic) được.

Xem lại ví dụ trong bài [**Tạo khung ứng dụng Express bằng epxres-generator**](https://chauit.com/tao-khung-ung-dung-expressjs-bang-express-generator/):

**routes/index.js**:

var express = require**(**'express'**)**;

var router = express.Router**()**;

/\* GET home page. \*/

router.get**(**'/', **function(**req, res, next**)** **{**

res.render**(**'index', **{** title: 'Express', name: 'Chauit' **})**;

**})**;

**module**.exports = router;

Đoạn code trên truyền 2 giá trị ‘Express’ và ‘Chauit’ bằng biến **title**, **name**cho file index.ejs

Sau đây mình sẽ làm lại ví dụ này, bằng cách bóc tách ra thành các file header.ejs, footer.ejs để code trong xịn và clean hơn, đồng thời thêm route “/user/:username” nhé.

Tạo ra 2 file header.ejs và footer.ejs trong thư mục views:

header.ejs:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**<%= title %>**</title>**

**<link** rel='stylesheet' href='/stylesheets/style.css' **/>**

**</head>**

**<body>**

footer.ejs:

**<footer>**

**<p>**Copyright © 2021 Chauit.com**</p>**

**</footer>**

**</body>**

**</html>**

Lúc này, trong file index.ejs, ta sẽ gọn như này:

<% include header%>

**<div** class="blurred-box"**>**

**<div** class="user-box"**>**

**<p** class="hello-user"**>**

Hello <%= name%>!

**</p>**

**<p>**Welcome to Express**</p>**

**</div>**

**</div>**

<% include footer%>

Giờ ta tạo thêm routes “user/:username” trong routes/index.js:

var express = require**(**'express'**)**;

var router = express.Router**()**;

/\* GET home page. \*/

router.get**(**'/', **function(**req, res, next**)** **{**

res.render**(**'index', **{** title: 'Express', name: 'Chauit' **})**;

**})**;

// show tên username bằng params :username

router.get**(**'/user/:username', **function(**req, res, next**)** **{**

const username = req.params.username;

res.render**(**'index', **{** title: 'Express', name: username **})**;

**})**;

**module**.exports = router;

Tạo thêm file user.ejs trong thư mục views:

**<**%- include header%**>**

**<**div **class**="blurred-box"**>**

**<**div **class**="user-box"**>**

**<**p **class**="hello-user"**>**

Hello user: **<**%= name%**>**!

**<**/p**>**

**<**p**>**Welcome to User Page**<**/p**>**

**<**/div**>**

**<**/div**>**

**<**%- include footer%**>**

1. **Model**
2. **NodeJS**

A screenshot of a video game

Description automatically generated with medium confidence

* Nodejs là một nền tảng (Platform) phát triển độc lập được xây dựng ở trên Javascript Runtime của Chrome mà chúng ta có thể xây dựng được các ứng dụng mạng một cách nhanh chóng và dễ dàng mở rộng.
* Phần Core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao.
* Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực.
* Nodejs chạy đa nền tảng phía Server, sử dụng kiến trúc hướng sự kiện Event-driven, cơ chế non-blocking I/O làm cho nó nhẹ và hiệu quả.
* Có thể chạy ứng dụng Nodejs ở bất kỳ đâu trên máy Mac – Window – Linux, hơn nữa cộng đồng Nodejs rất lớn và hoàn toàn miễn phí. Các bạn có thể thấy cộng đồng Nodejs lớn như thế nào tại đây, các package đều hoàn toàn free: <https://www.npmjs.com/>

1. **ExpressJS**

A picture containing logo

Description automatically generated

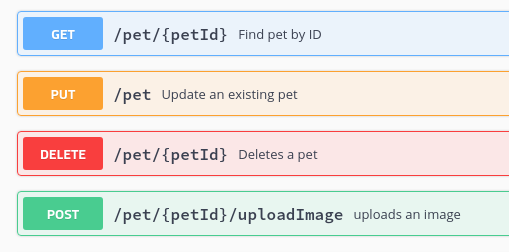
* Expressjs hay còn được viết là Express js, Express.js. Đây là một framework mã nguồn mở miễn phí cho Node.js. Express.js được sử dụng trong thiết kế và xây dựng các ứng dụng web một cách đơn giản và nhanh chóng
* Vì Express js chỉ yêu cầu ngôn ngữ lập trình Javascript nên việc xây dựng các ứng dụng web và API trở nên đơn giản hơn với các lập trình viên và nhà phát triển. Expressjs cũng là một khuôn khổ của Node.js do đó hầu hết các mã code đã được viết sẵn cho các lập trình viên có thể làm việc.
* Nhờ có Expressjs mà các nhà lập trình có thể dễ dàng tạo các ứng dụng 1 web, nhiều web hoặc kết hợp. Do có dung lượng khá nhẹ, Expressjs giúp cho việc tổ chức các ứng dụng web thành một kiến trúc MVC có tổ chức hơn. Để có thể sử dụng được mã nguồn này, chúng ta cần phải biết về Javascript và HTM
* Expressjs cũng là một phần của công nghệ giúp quản lý các ứng dụng web một cách dễ dàng hơn hay còn được gọi là ngăn xếp phần mềm MEAN. Nhờ có thư viện Javascript của Express js đã giúp cho các nhà lập trình xây dựng nên các ứng dụng web hiệu quả và nhanh chóng hơn. Expressjs cũng được sử dụng để nâng cao các chức năng của Node.js.
* Trên thực tế, nếu không sử dụng Express.js, bạn sẽ phải thực hiện rất nhiều bước lập trình phức tạp để xây dựng nên một API hiệu quả. Express js đã giúp cho việc lập trình trong Node.js trở nên dễ dàng hơn và có nhiều tính năng mới bổ sung.

*Những tính năng của Expressjs là gì?*

* Phát triển máy chủ nhanh chóng: Expressjs cung cấp nhiều tính năng dưới dạng các hàm để dễ dàng sử dụng ở bất kỳ đâu trong chương trình. Điều này đã loại bỏ nhu cầu viết mã từ đó tiết kiệm được thời gian.
* Phần mềm trung gian Middleware: Đây là phần mềm trung gian có quyền truy cập vào cơ sở dữ liệu, yêu cầu của khách hàng và những phần mềm trung gian khác. Phần mềm Middleware này chịu trách nhiệm chính cho việc tổ chức có hệ thống các chức năng của Express.js.
* Định tuyến - Routing: Express js cung cấp cơ chế định tuyến giúp duy trì trạng thái của website với sự trợ giúp của URL.
* Tạo mẫu - Templating: Các công cụ tạo khuôn mẫu được Express.js cung cấp cho phép các nhà xây dựng nội dung động trên các website bằng cách tạo dựng các mẫu HTML ở phía máy chủ.
* Gỡ lỗi - Debugging: Để phát triển thành công các ứng dụng web không thể thiết đi việc gỡ lỗi. Giờ đây với Expressjs việc gỡ lỗi đã trở nên dễ dàng hơn nhờ khả năng xác định chính xác các phần ứng dụng web có lỗi.
* Phân biệt giữa NodeJS và ExpressJS:
* Table

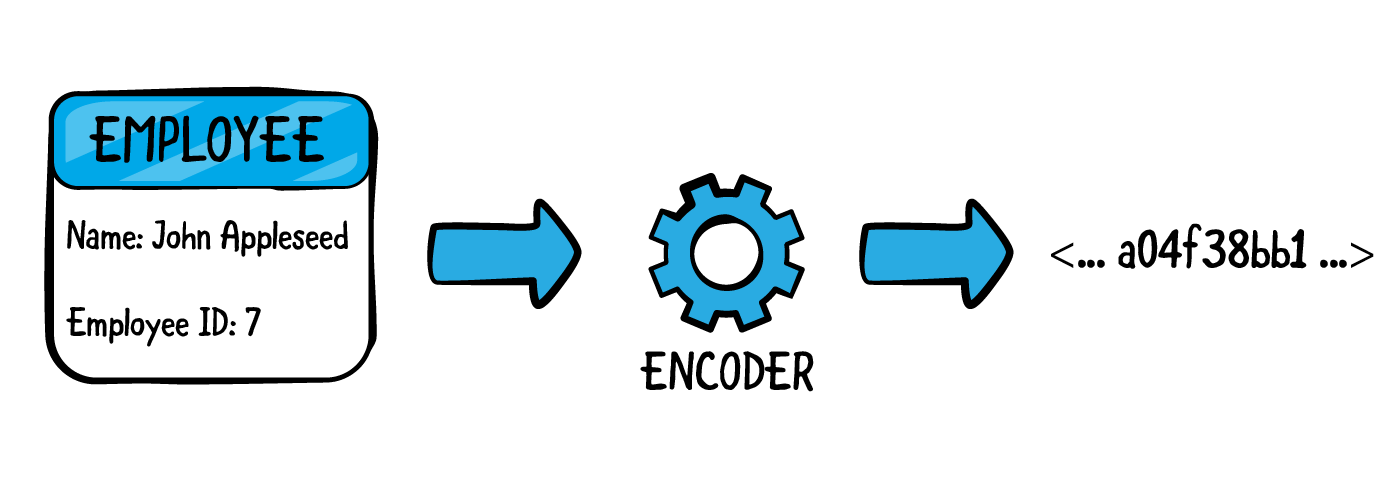
  Description automatically generated

1. **HTTP request methods**



* GET: được sử dụng để lấy thông tin từ sever theo URI đã cung cấp.
* HEAD: giống với GET nhưng response trả về không có body, chỉ có header
* POST: gửi thông tin tới sever thông qua các biểu mẫu http( đăng kí chả hạn..)
* PUT: ghi đè tất cả thông tin của đối tượng với những gì được gửi lên
* PATCH: ghi đè các thông tin được thay đổi của đối tượng.
* DELETE: xóa tài nguyên trên server.
* CONNECT: thiết lập một kết nối tới server theo URI.
* OPTIONS: mô tả các tùy chọn giao tiếp cho resource.
* TRACE: thực hiện một bài test loop - back theo đường dẫn đến resource.

1. **HTML Form Encoding**



* HTML Form Encoding xác định cách mã hóa dữ liệu gửi lên server khi submit form HTML. Giá trị này được xác định ở thuộc tính enctype trong thẻ mở form. Ta thường thấy 3 giá trị:
* *Application/x-www-form-urlencoded:* là kiểu mã hóa mặc định nếu thuộc tính enctype không có giá trị, đại diện cho URL Encoded Form. Tất cả các ký tự được mã hóa trước khi gửi (khoảng trắng được chuyển đổi thành ký hiệu "+" hoặc "%20" và các ký tự đặc biệt được chuyển đổi thành giá trị ASCII HEX).
* *Multipart/form-data*: đại diện cho Multipart form, kiểu mã hóa này được sử dụng khi người dùng muốn tải tệp dữ liệu lên.
* *text/plain*: Là một kiểu mới trong HTML 5, dữ liệu gửi lên mà ko có mã hóa.

1. **RESTful API**



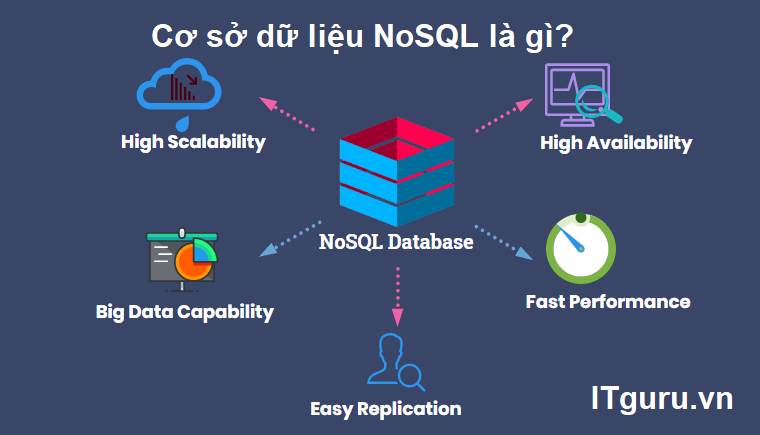
* RESTful API là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.
* 
* API (Application Programming Interface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.
* RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.
* Chức năng quan trọng nhất của REST là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một RESTful API.

1. **DATABASE**
2. **Big data**



* Big Data là thuật ngữ dùng để chỉ một tập hợp dữ liệu rất lớn và phức tạp đến nỗi những công cụ, ứng dụng xử lý dữ liệu truyền thống không thể thu thập, quản lý và xử lý dữ liệu trong một khoảng thời gian hợp lý.
* Những tập hợp dữ liệu lớn này có thể bao gồm các dữ liệu có cấu trúc (structured data), dữ liệu không cấu trúc (unstructured data) và dữ liệu nửa cấu trúc (semistructured data), mỗi tập hợp có chút khác biệt.
* Thông thường, Big Data có 3 điểm đặc trưng:
* Volume (Khối lượng): Lượng dữ liệu khổng lồ đang được lưu trữ.
* Velocity (Tốc độ): Tốc độ cực nhanh mà tại đó các luồng dữ liệu phải được xử lý và phân tích.
* Velocity (Tốc độ): Tốc độ cực nhanh mà tại đó các luồng dữ liệu phải được xử lý và phân tích.

1. **NoSQL**



* Cơ sở dữ liệu NoSQL sử dụng nhiều mô hình dữ liệu để truy cập và quản lý dữ liệu. Các loại cơ sở dữ liệu này được tối ưu hóa dành riêng cho các ứng dụng yêu cầu mô hình dữ liệu linh hoạt có lượng dữ liệu lớn và độ trễ thấp, có thể đạt được bằng cách giảm bớt một số hạn chế về tính nhất quán của dữ liệu của các cơ sở dữ liệu khác.
* Các loại cơ sở dữ liệu NoSQL:
* Khóa–giá trị: Cơ sở dữ liệu khóa–giá trị có khả năng phân mảnh cao và cho phép thay đổi quy mô theo chiều ngang ở các quy mô lớn mà các loại hình cơ sở dữ liệu khác không thể làm được. Mô hình dữ liệu khóa–giá trị được dùng cho trò chơi, công nghệ quảng cáo và đặc biệt thích hợp cho IoT. Amazon DynamoDB được thiết kế để có độ trễ ổn định chỉ vài mili giây cho khối lượng công việc thuộc quy mô bất kỳ. Hiệu năng ổn định này là lý do chính để di chuyển tính năng Stories của Snapchat (kể cả khối lượng công việc ghi lưu trữ lớn nhất của Snapchat) sang DynamoDB.
* Tài liệu: Trong mã ứng dụng, dữ liệu thường được biểu diễn dưới dạng một đối tượng hoặc văn bản dạng JSON vì đây là mô hình dữ liệu hiệu quả và trực quan cho các nhà phát triển. Cơ sở dữ liệu văn bản giúp nhà phát triển dễ dàng lưu trữ và truy vấn dữ liệu trong một cơ sở dữ liệu bằng cách sử dụng cùng một định dạng mô hình văn bản họ sử dụng trong mã ứng dụng của mình. Tính chất linh hoạt, bán cấu trúc và phân cấp của các văn bản và cơ sở dữ liệu văn bản cho phép chúng phát triển phù hợp với yêu cầu của ứng dụng. Mô hình văn bản phát huy hiệu quả với danh mục, hồ sơ người dùng và hệ thống quản lý nội dung, nơi từng văn bản là duy nhất và phát triển theo thời gian. Amazon DocumentDB (với khả năng tương thích MongoDB) và MongoDB là các cơ sở dữ liệu văn bản phổ biến cung cấp các API mạnh mẽ và trực quan để phát triển linh hoạt và lặp lại.
* Đồ thị: Mục đích của cơ sở dữ liệu đồ thị là giúp việc dựng và chạy ứng dụng hoạt động với các bộ dữ liệu có khả năng kết nối cao trở nên dễ dàng. Cơ sở dữ liệu đồ thị thường được sử dụng cho các đồ thị tri thức, mạng xã hội, công cụ đề xuất và phát hiện lừa đảo. Amazon Neptune là dịch vụ cơ sở dữ liệu đồ thị được quản lý đầy đủ. Neptune hỗ trợ cả mô hình Đồ thị thuộc tính lẫn Framework mô tả tài nguyên (RDF), cung cấp cho người dùng lựa chọn hai API đồ thị: TinkerPop và RDF/SPARQL. Các cơ sở dữ liệu đồ thị phổ biến gồm có Neo4j và Giraph.
* Trong bộ nhớ: Ứng dụng trò chơi và công nghệ quảng cáo đã sử dụng loại Cơ sở dữ liệu này trong nhiều trường hợp như bảng xếp hạng, kho lưu trữ phiên và phân tích thời gian thực yêu cầu thời gian phản hồi đạt mức micro giây và lưu lượng có thể đạt mức đỉnh điểm vào bất kỳ thời điểm nào. Amazon MemoryDB for Redis là một dịch vụ cơ sở dữ liệu trong bộ nhớ, bền, tương thích với Redis có chức năng đảm bảo độ trễ của thao tác đọc tính bằng micro giây, độ trễ của thao tác ghi chưa đến mười mili giây và độ bền trên nhiều vùng sẵn sàng. MemoryDB được xây dựng với mục đích đem lại độ bền và hiệu suất cực nhanh để bạn có thể sử dụng làm cơ sở dữ liệu chính cho các ứng dụng vi dịch vụ, hiện đại. Amazon ElastiCache là dịch vụ bộ nhớ đệm trong bộ nhớ được quản lý hoàn toàn, tương thích với cả Redis và Memcached để phục vụ khối lượng công việc có độ trễ thấp, thông lượng cao. Các khách hàng yêu cầu khả năng phản hồi theo thời gian thực cho ứng dụng của mình như Tinder phụ thuộc vào những kho lưu trữ dữ liệu trong bộ nhớ hơn là lưu trữ dữ liệu trên đĩa. Amazon DynamoDB Accelerator (DAX) là ví dụ khác về kho dữ liệu được xây dựng chuyên dụng. DAX giúp DynamoDB đọc nhanh hơn rất nhiều.
* Tìm kiếm: Nhiều ứng dụng xuất nhật ký để giúp nhà phát triển khắc phục sự cố. Amazon Elasticsearch Service (Amazon ES) là Cơ sở dữ liệu được xây dựng chuyên dụng để cung cấp khả năng ảo hóa và phân tích dữ liệu do máy tạo gần như theo thời gian thực bằng cách tạo chỉ mục, tập hợp và tìm kiếm nhật ký và số liệu có cấu trúc chưa hoàn chỉnh. Amazon ES còn là công cụ tìm kiếm mạnh mẽ và có hiệu năng cao dành cho các trường hợp sử dụng tìm kiếm toàn văn bản. Expedia đang sử dụng trên 150 miền Amazon ES, 30 TB dữ liệu và 30 tỷ văn bản cho nhiều chức năng đóng vai trò quan trọng, từ theo dõi và khắc phục sự cố trong khâu vận hành đến tối ưu hóa quá trình theo dấu và định giá chồng ứng dụng được phân phối.

1. **MongoDB**



* MongoDB là một chương trình cơ sở dữ liệu mã nguồn mở được thiết kế theo kiểu hướng đối tượng trong đó các bảng được cấu trúc một cách linh hoạt cho phép các dữ liệu lưu trên bảng không cần phải tuân theo một dạng cấu trúc nhất định nào. Chính do cấu trúc linh hoạt này nên MongoDB có thể được dùng để lưu trữ các dữ liệu có cấu trúc phức tạp và đa dạng và không cố định (hay còn gọi là Big Data).
* Một số đặc điểm của MongoDB:
* Kho lưu định hướng Document: Dữ liệu được lưu trong các tài liệu kiểu JSON.
* Lập chỉ mục trên bất kỳ thuộc tính nào.
* Các truy vấn đa dạng.
* Cập nhật nhanh hơn.

1. **Mongoose**

* Mongoose là một Object Document Mapper (ODM). Điều này có nghĩa là Mongoose cho phép bạn định nghĩa các object (đối tượng) với một schema được định nghĩa rõ ràng, được ánh xạ tới một MongoDB document.
* Mongoose cung cấp một số lượng đáng kinh ngạc các chức năng cho việc tạo ra và làm việc với các schema. Mongoose hiện có 8 SchemaTypes. Đó là:
* String
* Number
* Date
* Buffer
* Boolean
* Mixed
* ObjectId
* Array
* Mongoose là một thư viện JavaScript cho phép bạn định nghĩa các schema với dữ liệu được định kiểu rõ ràng. Khi một schema được định nghĩa, Mongoose cho phép bạn tạo một Model dựa trên một schema cụ thể. Model của Mongoose sau đó được ánh xạ tới một MongoDB document thông qua định nghĩa schema của Model.
* Khi bạn đã xác định các schema và model của mình, Mongoose chứa nhiều hàm khác nhau cho phép bạn xác thực tính hợp lệ, lưu, xóa và truy vấn dữ liệu của bạn bằng các hàm MongoDB phổ biến. Tôi sẽ nói về việc này nhiều hơn với các ví dụ code cụ thể.

1. **Kết quả đạt được**
2. **Mô hình Cơ Sở Dữ Liệu:**

Diagram

Description automatically generated

1. **Kết nối Database**

* **Code kết nối với Data mongoose**

const mongoose = require('mongoose')

const url = 'mongodb+srv://root:admin@cluster0.gsrso.mongodb.net/batdongsan?retryWrites=true&w=majority';

const connectionParams={

    useNewUrlParser: true, useUnifiedTopology: true

}

function connectDB(){

mongoose.connect(url,connectionParams)

    .then( () => {

        console.log('Connected to database ')

    })

    .catch( (err) => {

        console.error(`Error connecting to the database. \n${err}`);

    })

}

module.exports = connectDB;

* **Model Building.js kết nối với CSDL**

const mongoose = require("mongoose");

var buildingSchema = mongoose.Schema({

    name:String,

    rentarea:String,

    imagelink:String,

    street:String,

    districtid:String,

    ward:String,

    renttypeids:Array,

    note:String,

    managerphone:String,

    managername:String,

    rentprice:String,

    sellprice:String,

})

var building = mongoose.model("building",buildingSchema);

module.exports  = building;

* **Model Distict.js kết nối với CSDL**

const mongoose = require("mongoose");

var districtSchema = mongoose.Schema({

    name:String

})

var district = mongoose.model("district",districtSchema);

module.exports = district;

* **Model Rent Type kết nối với CSDL**

const mongoose = require("mongoose");

var renttypeSchema = mongoose.Schema({

    name:String

})

var renttype = mongoose.model("renttype",renttypeSchema);

module.exports = renttype;

1. **Giao diện chính của Website**

* **Gồm có các chức năng:**

+ CRUD Building

+ CRUD District

+ CRUD Rent Type

* **Code bên model của trang chủ**

const homePageController = require('../controllers/HomePageController');

const buildingRoute = require('../routes/building');

const districtRoute = require('../routes/district');

const renttypeRoute = require('../routes/renttype');

function route(app) {

    app.use('/building',buildingRoute);

    app.use('/district',districtRoute);

    app.use('/renttype',renttypeRoute);

    app.use('/',homePageController.index);

}

module.exports = route;

* **Code bên Controller thực hiện chức năng của trang chủ**

const building = require('../models/building');

const district = require('../models/district');

const renttype = require('../models/renttype');

class HomePageController{

    //[GET] / (HOME PAGE)

    index(req, res) {

        building.find(function (err, data) {

            if (err) throw err

            district.find(function (err, distrcits) {

                renttype.find(function (err, rentypes) {

                    var results = [];

                    var dis;

                    data.forEach(item => {

                        var rents = [];

                        distrcits.forEach(item2 => {

                            if (item2.\_id == item.districtid)

                                dis = item2.name;

                        })

                        item.renttypeids.forEach(item4 => {

                            rentypes.forEach(item3 => {

                                if (item3.\_id == item4) {

                                    rents.push(item3.name)

                                }

                            })

                        })

                        // var xuli = unique(rents)

                        var dataBuilding = {

                            "\_id": item.\_id,

                            "name": item.name,

                            "rentarea": item.rentarea,

                            "imagelink": item.imagelink,

                            "address": item.street + "-" + item.ward + "-" + dis,

                            "renttypes": rents,

                            "note": item.note,

                            "managername": item.managername,

                            "managerphone": item.managerphone,

                            "rentprice": item.rentprice,

                            "sellprice": item.sellprice

                        }

                        results.push(dataBuilding)

                    });

                    // res.json(results);

                    res.render('index', { listBuilding: results });

                })

            })

        })

    }

}

module.exports = new HomePageController;

* **Kết quả thực hiện được**

Graphical user interface

Description automatically generatedGraphical user interface, application

Description automatically generated

1. **Chức năng CRUD Building**

* **Thông tin Building gồm**: Name, Rent Area, Address, Rent Type, Manager Phone, Manager Name, Rent Price, Sell Price, Note, Image
* **Code thực hiện chức năng của Building :**

**+ Code bên Routers.**

const express = require('express');

const building = require('../controllers/BuildingController');

const router = express.Router();

//[GET] /building

router.get('/',building.index);

//[GET] buidling/insert

router.get('/insert',building.insertView);

//[GET] building/:id/update

router.get('/:id/update',building.updateView);

//[GET] building/:nameBuilding/search

router.get('/search',building.searchName)

//[POST] /building/insert

router.post('/insert',building.insertModel)

//[PUT] /building/:id/update

router.put('/:id/update',building.updateModel)

//[DELETE] /building/:id/delete

router.delete('/:id/delete',building.deleteModel)

module.exports = router;

**+ Code bên Controller thực hiện các chức năng chính.**

index(req, res) {

        if (req.query.name != "" && req.query.name != null && req.query.name != undefined) {

            building.find({ name:  new RegExp(req.query.name, 'i') }, function (err, data) {

                if (err) throw err

                district.find(function (err, distrcits) {

                    renttype.find(function (err, rentypes) {

                        var results = [];

                        var dis;

                        data.forEach(item => {

                            var rents = [];

                            distrcits.forEach(item2 => {

                                if (item2.\_id == item.districtid)

                                    dis = item2.name;

                            })

                            item.renttypeids.forEach(item4 => {

                                rentypes.forEach(item3 => {

                                    if (item3.\_id == item4) {

                                        rents.push(item3.name)

                                    }

                                })

                            })

                            // var xuli = unique(rents)

                            var dataBuilding = {

                                "\_id": item.\_id,

                                "name": item.name,

                                "rentarea": item.rentarea,

                                "imagelink": item.imagelink,

                                "address": item.street + "-" + item.ward + "-" + dis,

                                "renttypes": rents,

                                "note": item.note,

                                "managername": item.managername,

                                "managerphone": item.managerphone,

                                "rentprice": item.rentprice,

                                "sellprice": item.sellprice

                            }

                            results.push(dataBuilding)

                        });

                        // res.json(results);

                        res.render('building/index', { listBuilding: results });

                    })

                })

            })//

        } else {

            building.find(function (err, data) {

                if (err) throw err

                district.find(function (err, distrcits) {

                    renttype.find(function (err, rentypes) {

                        var results = [];

                        var dis;

                        data.forEach(item => {

                            var rents = [];

                            distrcits.forEach(item2 => {

                                if (item2.\_id == item.districtid)

                                    dis = item2.name;

                            })

                            item.renttypeids.forEach(item4 => {

                                rentypes.forEach(item3 => {

                                    if (item3.\_id == item4) {

                                        rents.push(item3.name)

                                    }

                                })

                            })

                            // var xuli = unique(rents)

                            var dataBuilding = {

                                "\_id": item.\_id,

                                "name": item.name,

                                "rentarea": item.rentarea,

                                "imagelink": item.imagelink,

                                "address": item.street + "-" + item.ward + "-" + dis,

                                "renttypes": rents,

                                "note": item.note,

                                "managername": item.managername,

                                "managerphone": item.managerphone,

                                "rentprice": item.rentprice,

                                "sellprice": item.sellprice

                            }

                            results.push(dataBuilding)

                        });

                        // res.json(results);

                        res.render('building/index', { listBuilding: results });

                    })

                })

            })//

        }

    }

**+ Kết quả thực hiện được**

Text

Description automatically generated with low confidence

**+ Chức năng thêm Building**

//[POST] /building/insert

    insertModel(req, res) {

        var dataInsert = req.body;

        var dataBuilding = {

            "name": dataInsert.name,

            "rentarea": dataInsert.rentarea,

            "imagelink": dataInsert.imagelink,

            "street": dataInsert.street,

            "districtid": dataInsert.districtid,

            "ward": dataInsert.ward,

            "renttypeids": dataInsert.rentypes,

            "note": dataInsert.note,

            "managername": dataInsert.managername,

            "managerphone": dataInsert.managerphone,

            "rentprice": dataInsert.rentprice,

            "sellprice": dataInsert.sellprice,

        }

        building.create(dataBuilding, function (err, result) {

            if (err)

                res.send("INSERT FAILURE");

            else {

                res.json({

                    "status": "success"

                });

            }

        });

    }

**+ Kết quả thực hiện được**

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

**+ Code thực hiện chức năng update**

 //[GET] building/:id/update

    updateView(req, res) {

        var id = req.params.id;

        //   var dataBuilding,dataDictrict

        building.findOne({ \_id: id }, function (err, dataBuilding) {

            if (err)

                res.send(err);

            else {

                district.find(function (err, dataDictrict) {

                    renttype.find(function (err, dataRenttype) {

                        var Data = {

                            building: dataBuilding,

                            district: dataDictrict,

                            renttype: dataRenttype

                        }

                        res.render('Building/editBuilding', { data: Data });

                    })

                })

            }

        })

    }

**+ Kết quả thực hiện được**

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

**+ Code thực hiện chức năng tiềm kiếm Building**

//[GET] /search/:nameBuilding

    searchName(req, res) {

        var nameBuilding = req.query.nameBuilding.toString();

        building.find({ name: nameBuilding }, function (err, data) {

            res.render('/buidling/searchBuilding', { list: data });

        });

    }

**+ Kết quả thực hiện được khí tìm kiếm tên tòa nha 450**

Graphical user interface, application, Word

Description automatically generated

**+ Code thực hiện chức năng Delete Building**

  //[DELETE] /building/:id/delete

    deleteModel(req, res) {

        var id = req.params.id;

        building.deleteOne({ \_id: id }, function (err, data) {

            if (err)

                res.send(err);

            else {

                res.status(200).json("oke");

            }

        })

    }

**+ Kết quả thực hiện được xóa thành công tòa nhà 450.**

1. **Chức năng CRUD District**

**+ Thông tin District gồm:** Name

**+ Code bên Router**

const district = require("../controllers/districtController");

const express = require('express');

const router =express.Router();

//[GET] /dictrict

router.get('/', district.index);

//[GET] /dictrict/insert

router.get('/insert',district.insertView);

//[GET] /dictrict/:id/update

router.get('/:id/update',district.updateView);

//[GET] /district/search/:nameDistrict

router.get('/search/:id',district.search);

//[POST] /district/insert

router.post('/insert',district.insertModel);

//[PUT] /district/:id/update

router.put('/:id/update',district.updateModel);

//[DELETE] /district/:id/delete

router.delete('/:id/delete',district.deleteModel);

module.exports = router;

**+ Chức năng thêm District**

   //[POST] /district/:id/insert

    insertModel(req,res){

        var data = req.body;

        var dataDictrict = {

            name : data.name

        }

        district.create(dataDictrict,function(err,data){

            if(err)

                res.send(err);

            else

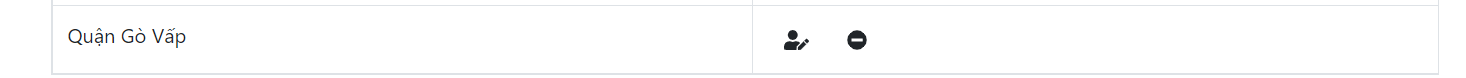
                 res.json("oke");

        })

    }

**+ Kết quả thực hiện**

Graphical user interface, application

Description automatically generated

**+ Code chức năng update District**

 //[PUT] /district/:id/update

    updateModel(req,res){

        var idDistrict = req.params.id;

        var dataBody = req.body;

        var dataDistrict = {

            name: dataBody.name

        }

        district.updateOne({\_id:idDistrict},dataDistrict,function(err,data){

            if(err)

                res.send(err);

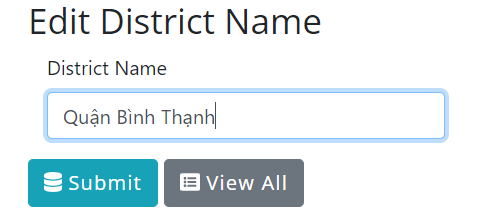
            else

                 res.json("oke");

        })

    }

**+Kết quả thực hiện sửa Quận Gò Vấp thành Quận Bình Thạnh**



**+ Chức năng Delete District**

 //[DELETE] /district/:id/delete

    deleteModel(req,res){

        var id = req.params.id;

        district.deleteOne({\_id:id},function(err,data){

            if(err)

                res.send(err);

            else{

                 res.json("oke");

            }

        })

    }

**+ Kết quả thức hiện được xóa thành công Quận Bình Thạnh**

1. **Chức năng CRUD Rent Type**

**+ Thông tin District gồm:** Name

**+ Code bên Router**

const express = require('express');

const router = express.Router();

const renttype = require('../controllers/renttypeController');

//[GET] /renttype

router.get('/',renttype.index);

//[GET] /renttype/insert

router.get('/insert',renttype.insertView);

//[GET] /:id/search

router.get('/search',renttype.searchName);

//[POST] /renttype/insert

router.post('/insert',renttype.insertModel);

//[GET]  /renttype/:id/update

router.get('/:id/update',renttype.updateView);

//[PUT] /renttype/:id/update

router.put('/:id/update',renttype.updateModel);

//[DELETE] /renttype/:id/delete

router.delete('/:id/delete',renttype.deleteModel);

module.exports = router;

**+ Chức năng thêm Rent Type**

  //[POST] /renttype/insert

    insertModel(req, res) {

        var data = req.body;

        var dataRenttype = {

            name: data.name

        }

        renttype.create(dataRenttype, function (err) {

            if (err)

                res.send(err);

            else {

                res.json("oke");

            }

        })

    }

**+ Kết quả thực hiện**

Application

Description automatically generated with medium confidence



**+ Code chức năng update Rent Type**

//[GET] /renttype/:id/update

    updateView(req, res) {

        var idRenttype = req.params.id;

        renttype.findOne({ \_id: idRenttype }, function (err, data) {

            if (err)

                res.send(err);

            else {

                res.render('Renttype/update', { data: data });

            }

        })

    }

//[PUT] /renttype/:id/update

    updateModel(req, res) {

        var idRenttype = req.params.id;

        var dataBody = req.body;

        var data = {

            name: dataBody.name

        }

        renttype.updateOne({ \_id: idRenttype }, data, function (err, data) {

            if (err)

                res.send(err);

            else {

                res.json("oke");

            }

        })

    }

**+Kết quả thực hiện sửa Nhà xe thành Nhà Ôtô**

Graphical user interface, text, application, chat or text message

Description automatically generated



**+ Chức năng Delete Rent Type**

 //[DELETE] /renttype/:id/delete

    deleteModel(req, res) {

        var id = req.params.id;

        renttype.deleteOne({ \_id: id }, function (err, data) {

            if (err)

                res.send(err);

            else {

                res.json("oke");;

            }

        })

    }

}

**+ Kết quả thức hiện được xóa thành công Nhà Ô tô**

1. **Nhận xét và cải tiến**