|  |
| --- |
| **node** |
| Để chạy file index.js  Tại terminal gõ:  node index.js : sẽ chạy file index.js |
| **npm** |
| **npm init** : khởi tạo package.json  **npm start** : chạy đoạn “start” nằm trong “script” trong file package.json  **npm i express** : tải thư viện express về node\_modul |
| **nodemon : thư viện giúp nhận biết các thay đổi rồi up lên server** |
| Khi không sài nodemon mỗi lền bạn thay đổi code phải tắt server rồi chạy lại thì nó mới nhận thay đổi đó, với nodemon bạn không cần làm vậy nữa  Tải nodemon  npm i nodemon --save-dev : save nodemon vào devDependencies  Vào package.json 🡺 “script” : { thêm đoạn mã sau  "start":"nodemon --inspect index.js",  Tại terminal gõ lệnh:  npm start |
| **Add git repository** |
| Tại terminal:  git init : khởi tạo git |
| **Morgan** |
| Giúp quan sát các Request được gửi từ phía client  Tại terminal : npm i morgan --save-dev |
| **Template engine** |
| app.get('/', (*req*, *res*) => {  *res*.send(`<h1>Hello World</h1>`)  })  Ta dùng Template engine để thay cho đoạn mã HTML trong phần res.send() |
| **Static files & SASS** |
| Cấu hình dự án sử dụng được các file tĩnh : img , css ,  BIết cấu hình CSS processor: để sử dụng scss  **1. Static file:**  Trong file src tạo file public ( lưu các file tĩnh trong này)  Trong file index.js:  app.use( express.static( path.join(\_\_dirname, 'public') )  );  // lúc này khi vào  <http://localhost:3000/img/logo.png>  nó sẽ truy cập vào file tĩnh img/logo.png nằm trong file public  **2. Node Sass**  cài đặt:npm i node-sass --save-dev  **Trong file resources : tạo 1 file scss/app.scss**  **Trong file public tạo 1 file css**  Vì trình duyệt không đọc được file app.scss nên ta phải chuyển nó thành file css nằm trong thư mục css rồi mới đẩy lên trình duyệt.  Trong file package.json :  Thêm vào “script”:  "watch": "node-sass src/resources/scss/ --output src/public/css/",  // lúc này khi gõ : npm run watch nó sẽ tự động chuyển file trong scss 🡺 app.css và lưu vào file css  "watch": "node-sass --watch src/resources/scss/app.scss src/public/css/app.css",  -- Thêm option : --watch nó sẽ tự lắng nghe sự thay đổi trên app.scss để thay đổi vào file app.css mà không cần phải gõ npm run watch nhiều lần. |
| **Bootstrap 4** |
| - doc : <https://getbootstrap.com/docs/4.0/getting-started/download/>  Chèn link vào trước thẻ link css  <link *href=*"https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" *rel=*"stylesheet" *integrity=*"sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" *crossorigin=*"anonymous">  Chèn script sau thẻ <body>  <!-- jQuery library -->  <script *src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  <!-- Popper JS -->  <script *src=*"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>  <!-- Latest compiled JavaScript -->  <script *src=*"https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script> |
| **Basic routing** |
| URL:đường dẫn và URI: đường dẫn cố định  vd: <https://fullstack.edu.vn> là URI  <https://fullstack.edu.vn/learning/reactjs?id=2744> là URL |
| **HTTP: Get method** |
| Mặc định trình duyệt khi truy cập qua thanh địa chỉ , ta sẽ tạo một yêu cầu với phương thức là GET lên server. |
| **Query parameters** |
| Truyền dữ liệu qua chính trình duyệt của bạn  Giống với < form method= “get”>  vd: [http://localhost:3000/search**?**q=f8](http://localhost:3000/search?q=f8) laptrinh**&**author=sondang  - truyền các query parameter sau dấu ? và cách nhau bằng dấu &  Sử dụng debug cua nodeJS đặt breakpoint tại vị trí sau:  app.get('/search', (*req*, *res*) => {  *res*.render('search');  });  Tại tham số req ta có thể nhận được query parameter qua phương thức  req.query = {  q : ‘f8 laptrinh,  author : “sondang”  } |
| **Form default behavior** |
| Tìm hiểu hành vi mặc định của Form trong HTML  <form *method=*"get" *action=*"">  method: Chọn Request method , mặc định là get  action: Chọn RequestURL , mặc định là URL hiện tại  vd: <  - Tại URL: <http://localhost:3000/search>  ẤN SUBMIT tại form :  <form *method=*"get" *action=*"">  🡺 Thực hiện Request method: GET tại URLrequest:  <http://localhost:3000/search>?....query\_parameters....  - Tại URL: <http://localhost:3000/search>  ẤN SUBMIT tại form :  <form *method=*"post" *action=*"/news">  🡺 Thực hiện Request method: POST tại URLrequest: <http://localhost:3000/news>  1. form method= “get” : mặc định  Sau khi nhập vào ô input và ấn submit ta sẽ tạo ra một URLrequest có đính kèm Query Parameter và tạo một HTTP Request method là GET tại chính URL diễn ra hành vi này  vd: [http://localhost:3000/search?q=*[your-text*](http://localhost:3000/search?q=%5byour-text)*]...*   <form *method=*"GET">          <div *class=*"form-group">              <label *for=*"search-input">Tử khoá</label>              <input *type=*"text" *name=*"q" *class=*"form-control" *id*="search-input" *aria-describedby=*"emailHelp" *placeholder=*"Nhập từ khoá">          </div>            <button *type=*"submit" *class=*"btn btn-primary">Tìm kiếm</button>      </form>  2. form method= “post”  Khi ấn submit , sẽ gửi một Request Method là post lên URL  vd: Khi ấn submit tại form action= “/search” sẽ gọi lên server thực hiện lệnh:  app.post('/search', (*req*, *res*) => {  *res*.render('search');  });  Các nội dung được gửi đi sẽ không đính kèm lên URL mà nằm trong mục Form Data vẫn dưới dạng key value .  Sau khi ấn submit , dữ liệu từ Browser 🡺 Middle ware 🡺 Controller  Do đó ta không nhận trực tiếp được trong phần **req**  Để sử dụng Middle ware cho form method=”post” , ta thêm lệnh sau:  app.use(express.urlencoded());  Lúc này ta có thể truy xuất nội dung trong Form data trong mục req bằng req.body  app.post('/search', (*req*, *res*) => {   console.log(*req*.body) } |
| **Mô hình MVC** |
| Model - View - Controller  View: Thành phần chứa các file view như HTML, CSS  Controller: Thành phần chứa file xử lý  như JS, SCSS,…  Model: Chứa data base  Giải nghĩa Mô hình:  1. **Browser** ( Hay còn gọi là phía client theo mô hình client - server)  Browser sẽ tạo HTTP request gồm **RequestUrl** và **Request** **Method** gửi lên Web Server  Web Server chạy khi ta bắt đầu gõ lệnh:  npm start  Chính xác hơn là gõ lệnh: Node src/index.js  Do chúng ta dùng thư viện Express và Nodemon nên ta gõ lênh start để chạy đoạn mã sau:  "start": "nodemon --inspect src/index.js",  **2. Web Server** dựa vào phương thức HTTP request mà quyết định sẽ làm gì  vd:  <form *method=*"get" *action=*"/search">  Khi ấn Submit ta đã tạo một HTTP request với **Request** **Method = get và**  **RequestUrl** [http://localhost:3000/search? q=*[your-text*](http://localhost:3000/search?%20q=%5byour-text)]  . Lúc này Wed Server sẽ chạy đoạn script sau:  app.get('/search', (*req*, *res*) => {  *res*.render('search');  });  - Trong đó : **app.get(‘/search’)** chính là **ROUTER** , đường dẫn được xác định dựa vào HTTP request ở trên  - Sau đó đoạn mã bên trong được thực thi: **( req, res) => res.render(‘search’)**  Đoạn mã này chính là thành phần **Controller**  - Phần **Dispatcher** : là phần xác định trung gian nếu đúng **ROUTER** thì phần **Controller** mới được thực hiện  **3. Controller**  **( req, res) => res.render(‘search’)**  Controller gửi yêu cần **render** ra thư mục **search cho VIEW**  **4. VIEW**  Khi VIEW nhận được yêu cầu sẽ trả lại toàn bộ thư mục search cho Controller,  Rồi Controler sẽ render ra WebServer gửi ra Browsers |
| **Install Mongodb** |
|  |
| **Prettier- Code formatter** |
| **- Thư viện Prettier :** format lại code cho đẹp hơn  **- lint-staged** : Giúp chạy một command xử lý các code đã post lên git  **- husky:** cung cấp Git Hook để mỗi lần chạy **commit** sẽ chạy **lint-staged** để chạy **prettier**  Cài đặt: npm i prettier lint-staged husky --save-dev  Để sử dụng Prettier :  CLI : comman line interface  **prettier [option] [file/ dir/ glob … ]**  **--write** // ghi đè file của các bạn cho ngay hàng thằng lối  **--single-quote** // đổi dấu “ thành dấu ‘  **--trailing-comma** // thêm dấu , vào cuối Array hoặc Object  vd: prettier --write --single-quote --trailing-comma all ‘src/\*\*/\*.{js,scss,json}’  /\*\* : đi vào các thư mục bên trong  /\* : chọn tất cả thư mục có đuôi là .(js , scss hoặc json)  Vào thư mục packaged.json thêm vào phần script  "beautiful": "prettier --single-quote --trailing-comma all --write   \"src/\*\*/\*.{js,json,scss}\" ",  Vào terminal gõ: npm run beautiful để thực hiện đoạn mã trên  🡺 **Để sử dụng lint-staged**  Chèn đoạn sau trong packaged.json   "lint-staged" : {        "src/\*\*/\*.{js,json,scss}" : "prettier --single-quote --trailing-comma all --tab-width 4 --write   \"src/\*\*/\*.{js,json,scss}\" "    },  Và đổi đoạn script : “beatiful” trở thành  "beautiful" : "lint-staged"  Khi đó mỗi lần chạy cmd: npm run beautiful , nó sẽ thực hiện **lint-staged** là kiểm tra các file đã được add vào git hay chưa, nếu có nó sẽ thực hiện đoạn mã Prettier bên trong.  **🡺 Để sử dụng Husky**  **Thêm đoạn mã sau vào file packaged.js**  "husky":{      "hooks" : {        "pre-commit" : "lint-staged"      }   },  Khi đó mỗi lần commit code lên git phần lint-staged sẽ được chạy mà không cần phải chạy npm run beautiful nữa |
| **[MVC] model** |
| // Hướng dẫn Mongo DB compass  Là một client để connect tới server Mongodb  **- Creat Database**  Database Name: f8\_education\_dev  Collection Name: courses  - **courses**  ADD Data: 🡺 Insert Document  \_id: ObjectId("617a750854cc3fb5345a82e8"), // id do Mongodb tự tạo ra  **name** : "Xây dụng web với Nodejs & Express",  **description** : "Xây dựng website trong thực tế với Nodejs & ExpressJS với cách chia sẻ chi tiết, tận tâm , dễ hiểu và chất giọng giàu sức sống của người chia sẻ",  **image** : <https://cdn.fullstack.edu.vn/f8-learning/courses/6.png>  **- Install mongoose**  Mongoose : Thằng này đứng giữa như cầu nối giữa **Nodejs** và **Mongodb**  Mongoose là tool hỗ trợ mô hình hoá Mongodb **(document database) ,** làm việc với môi trường bất đồng bộ như JavaScript.  **npm install mongoose**  **- Connect to DB :**  **Tạo forder : src/config/db/index.js**  const *mongoose* = require('mongoose');  async function connect() {     try {          await *mongoose*.connect("mongodb://localhost:27017/f8\_education\_dev);          console.log('connect successful');     }     catch (error) {      console.log('connect tạch');     }  };  *module*.*exports* = { connect };  **Tại mục: src/index.js**  const db = require('./config/db/index.js');  // Connect to db  db.connect();  **- Creat model**  **Tạo forder app/models/Course.js:**  const *mongoose* = require('mongoose');  const Schema = *mongoose*.Schema;  const Course = **new** Schema({      name: { type: *String*},      description: { type: *String*},      image: { type: *String*},      createAt: {type: *Date*, default: *Date*.now},      updatedAt: {type: *Date*, default: *Date*.now},  });  *module*.*exports* = *mongoose*.model('Course', Course);  Tại thư mục SiteController.js , ta truyền model  const Course = require('../models/Course.js');  class SiteController {      //[GET] /      home(*req*, *res* , *next*) {            Course.find({})              .then( *courses* => *res*.render('home', {courses: *courses*}) )              .catch( *error* => next(*error*) )      }  Tại thư mục home.hbs ta nhận được dữ liệu qua dạng Object {{coures}} |
| **Course Detail page** |
| Khi truyền vào web dạng: <http://localhost:3000/courses/reactjs>  route.get('/:slug', courseController.course);   show(*req*, *res* , *next*) {          Course.find({})              .then( *courses* => {                  console.log(*req*.params.slug);  *res*.render('course', {courses})              })              .catch( *error* => next(*error*) )   }  ta có thể nhận được biến thêm sau bằng  *req*.params.slug  Do đó ta có thể dùng Course.findOne( { key : value } ) để tìm khoá học cần render ra  show(*req*, *res*, *next*) {          Course.findOne({ slug: *req*.params.slug })              .then((*course*) => *res*.render('courses/show', {course}))              .catch(*next*);   } |
| **(CRUD) Creat New Course** |
| Tạo phần nhập dữ liệu vào Mongodb  Tạo một form với phương thức là post  Tại thư mục : courses.js  route.get('/create', courseController.create);  route.post('/store', courseController.store);  Với phương thức get : /creat dẫn tới trang nhập thêm thông tin mới   <form  *method=*"POST" *action=*"http://localhost:3000/courses/store">  Khi ấn submit form sẽ gửi yêu cầu với phương thức POST tại url trên  Tại thư mục: CourseController.js   store(*req*, *res*, *next*){          Course.create(*req*.body)              .then(() => *res*.redirect('/'))              .catch((*error*) => next(*error*));      }  Ta nhận các thông tin gửi kèm ở form qua req.body  -- Course.creat( { } ) : Tạo dữ liệu mới trên data  res.redirect( ) : Điều hướng trang web |
| **moongo slug generator** |
| Hỗ trợ tự tạo slug bằng cách chuyển đổi Name thành dạng chữ không dấu cách nhau dấu gạch ngang  Cài đặt: npm install mongoose-slug-generator --save  Tại thư mục: Course.js  Thêm slug  const slug = require('mongoose-slug-generator');  *mongoose*.plugin(slug);  tại slug:  slug: { type: *String* , slug: 'name' , unique: true },  Slug sẽ được tự động tạo từ name và thêm unique để nếu có slug trùng nó sẽ thêm ký tự đặc biệt vào ,  // khi kiểm tra packed-lock.json sẽ thấy mongoose-slug-generator có một require là shortid , dùng để tạo id ngẫu nhiên gắn vào slug nếu thấy trùng nhau |
| **[CRUD] Update Course** |
| replace(*req*, *res*, *next*){          Course.updateOne({\_id: *req*.params.id} , *req*.body)                .then(() => *res*.redirect('/courses/update'))      }  Truyền dữ liệu update bằng form với method = post  để nhận mọi dữ liệu vào req.body  Dùng \_id để làm dấu hiệu nhận biết |
| **[CRUD] Delete Course** |
| delete(*req*, *res*, *next*){          Course.deleteOne({slug: *req*.params.slug})                .then(() => *res*.redirect('/courses/update'))      }  // Nên tạo một modal xác nhận xoá để tránh hưởng ngưởi dùng ấn nhầm á |
| **Soft Delete** |
| Xoá mềm: Chỉ ẩn đi nội dung trên UI nhưng vẫn còn dữ liệu trên database  Vd: Sử dụng thùng rác, đưa thư mục muốn xoá vào thùng rác  - Tạo một trường dữ liệu trong database:  vd: deteledAt : null ( data-type: date)  - Khi **xoá** sẽ tự động ghi đè lại trường dữ liệu này  - Khi **khôi phục** sẽ ghi đè giá trị Null lại trường này  - Do đó , khi render ra UI chỉ render những dữ liệu có giá trị **deteledAt : null**  **Một soft Delete phải thực hiện được 3 chức năng cơ bản:**  **- Delete (soft) - xoá mềm**  **- Restore - Khôi phục**  **- Force delete - xoá cứng**  **🡺 Sử dụng plugin : npm i mongoose-delete để hỗ trợ**  const mongooseDelete = require('mongoose-delete');  Course.plugin( mongooseDelete, {deleteAt: true , overrideMethods: 'all' });  -- Do có tính chất override nên phương thức **find()** của mongoose sẽ tự động loại những thằng có trường dữ liệu  Sử dụng các phương thức hỗ trợ sau của mongoose-delete  **delete( )** - Thêm trường deleted: true vào  **findDelete( )** - Tìm những dữ liệu có deleted: true  **restore( { } )** - Gán lại deleted : false cho dữ liệu cần phục hồi |
| **SelecAll with Checkbox** |
| Nút chọn tất cả:  - Tạo các checkItem và checkAll  - Mỗi checkItem mang một \_id của dữ liệu  - Sử dụng cú pháp của mongoose để thêm phương thức cho nhiều dữ liệu cùng lúc**: { $in: *myList* } // myList ở dạng mảng**  **vd: Course.delete( { \_id: { $in: *myDeleteList* } )** |
| **Middleware với ExpressJS** |
| Là các funtion chen giữa và kết thúc bằng next() để thực hiện function tiếp theo:  vd:  Course.findOne({slug: *req*.params.slug })              .lean()              .then((*course*) => *res*.render('courses/edit', {course}))  finOne hay lean đều được gọi là middleware |
| **Chức năng sắp xếp:** |
| 1. Tạo file controlers/middlewares/ SortMiddleware.js  *module*.*exports* = function SortMiddleware(*req* , *res* ,  *next*){    *res*.locals.\_sort = {          enabled : false,          type : 'default',          colum: 'default',      }      // phân tích HTTP request để nhận …?\_sort&colum=…..&type=…..      if(*req*.query.hasOwnProperty('\_sort')){  *Object*.assign(   *res*.locals.\_sort , {              enabled: true,              type : *req*.query.type,              colum: *req*.query.colum,          })      };      next()  }  Với phương thức : res.locals.\_sort ta tạo ra một biến cục bộ và biến này có thể truy xuất ở các function tiếp theo hoặc trong handlebar {{\_sort}}  2. tại file index.js gọi middleware ra để sử dụng  const SortMiddleware  = require('./app/middlewares/SortMiddleware');  app.use(SortMiddleware)  Lúc này tất cả các hoạt động của app sẽ đều đi qua middleware ta vừa tạo  3. Tại CourseController.js ta muốn sử dụng middleware này ở phần update      update(*req*, *res*, *next*){          let courseQuery = Course.find({});          const \_sort = *res*.locals.\_sort;          if(\_sort.enabled) {              courseQuery = Course.find({}).sort({                  [\_sort.colum] : \_sort.type              })          }            courseQuery              .lean()              .then((*courses*) => *res*.render('courses/update', {courses}))              .catch(*next*);  Dùng phương thức sort( ‘key’ : option) để sắp xếp dữ liệu của Courses trước khi đưa nó xuống các hàm còn lại |
| **npm** |
| - Project Scope : Phạm vị dự án  - Global Scope : Phạm vi máy tính , xài đi xài lại được |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |