**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**QUẢN LÝ DỰ ÁN PHẦN MỀM**

**NHÓM 2**

**WEBSITE ĐĂNG TIN TUYỂN DỤNG VÀ TÌM VIỆC LÀM THÊM**

**Proof Of Concept**

Giáo viên LT : **Ngô Huy Biên**

Giáo viên TH : **Ngô Ngọc Đăng Khoa**

Tháng 9/2017

**Bảng ghi nhận thay đổi tài liệu**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ngày** | **Phiên bản** | **Mô tả** | **Tác giả** |
| 18/09/2017 | 1.0 | Soạn thảo tài liệu | Bùi Duy Khôi, Đoàn Văn Huy |
|  |  |  |  |
|  |  |  |  |

**NỘI DUNG TÀI LIỆU**

[I. GIỚI THIỆU 1](#_Toc493537737)

[A. MỤC ĐÍCH CỦA TÀI LIỆU 1](#_Toc493537738)

[B. ĐỐI TƯỢNG CỦA TÀI LIỆU 1](#_Toc493537739)

[C. TÁC GIẢ 1](#_Toc493537740)

[II. KHỞI ĐỘNG ỨNG DỤNG 1](#_Toc493537741)

[III. MÔ TẢ COMPONENT 3](#_Toc493537742)

[A. FRONTEND 3](#_Toc493537743)

[B. BACKEND 4](#_Toc493537744)

# GIỚI THIỆU

## MỤC ĐÍCH CỦA TÀI LIỆU

* Nhằm hướng người đọc có cái nhìn cụ thể về ứng dụng: cách thức khởi động, các thành phần quan trọng của ứng dụng.
* Xét về ngữ cảnh ứng dụng, tài liệu sẽ mô tả cách khởi động webserver, các thành phần quan trọng của frontend và backend thuộc về ứng dụng.

## ĐỐI TƯỢNG CỦA TÀI LIỆU

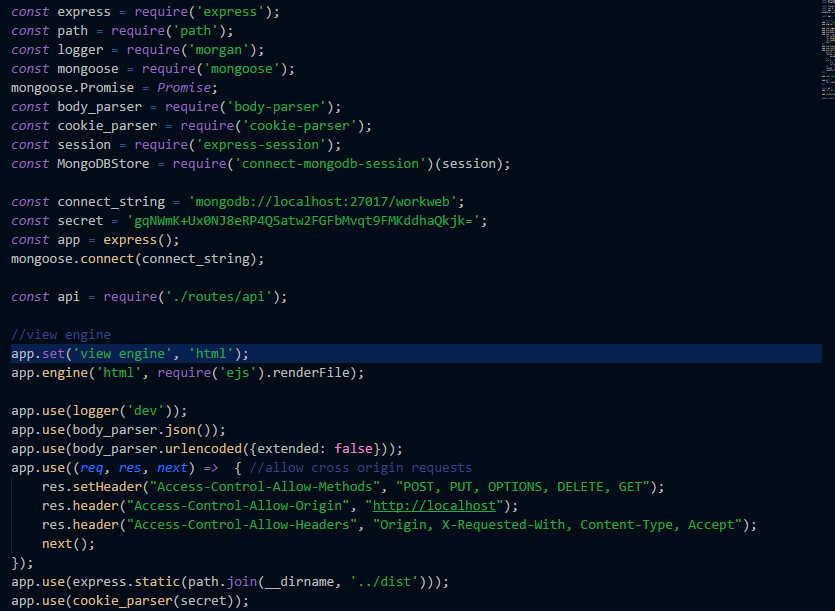
* Thầy **Ngô Huy Biên**
* Thầy **Ngô Ngọc Đăng Khoa**
* Những ai có mong muốn tìm hiểu về quá trình làm nên ứng dụng

## TÁC GIẢ

* Đoàn Văn Huy - 1412195
* Bùi Duy Khôi - 1412258

# KHỞI ĐỘNG ỨNG DỤNG

* Để có được repo của ứng dụng, sử dụng git clone với đường dẫn <https://github.com/Neoreul/qldapm.git>
* Ứng dụng cần có những thành phần (dependencies) sau để khởi chạy:
  + **Nodejs**: nền tảng webserver của ứng dụng
  + **npm – Node Package Manager**: bộ quản lý các gói thư viện nguồn mở javascript
  + **bower**: bộ quản lý các gói thư viện cho web client
* Truy cập vào thư mục gốc của ứng dụng, gõ lệnh: **npm install** để tải các thư viện cần thiết cho server side. Sau khi xong, các gói thư viện sẽ được lưu trong thư mục **node\_modules.** Các gói thư viện này được định nghĩa trong **package.json.**
* Tiếp tục gõ lệnh: **bower install** để tải các thư viện cần thiết cho client side. Sau khi xong, các gói thư viện sẽ được lưu trong thư mục **app/components.** Các gói thư viện này được định nghĩa trong **bower.json.**
* Trước khi chạy, cần phải build ứng dụng với câu lệnh: **npm run build.** Sau khi xong, ta sẽ được một thư mục dist chứa toàn bộ tài nguyên cần thiết của ứng dụng.
* Khởi chạy server bằng lệnh: **npm start**
* Web client sẽ truy cập đến localhost tại cổng 3000 để bắt đầu sử dụng ứng dụng
* Mã nguồn khởi động server (**server/server.js**):

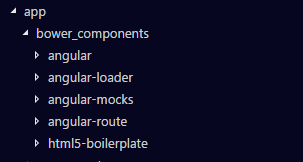
****

****

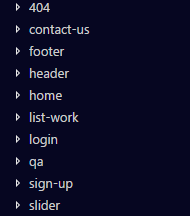
# MÔ TẢ COMPONENT

## FRONTEND

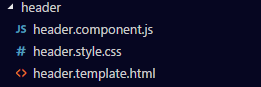
* Web client được cài đặt dưới dạng **Single Page Application** sử dụng thư viện **AngularJs (Angular 1)**



* angular: thư viện chính để cài đặt **Single Page Application** cho web client
* angular-route: định tuyến các file html theo url xác định
* html5-boilerplate: xây dựng các template html
* Các thành phần của web (**app/components/web-frontend**):



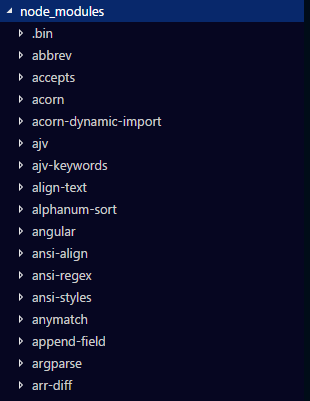
* **404**: trang báo lỗi đường dẫn không tồn tại
* **contact-us**: trang hiển thị thông tin liên lạc giữa các thành viên trong nhóm
* **footer**: footer của web
* **header**: header của web – chứa các chỉ mục chuyển hướng đến các chức năng của ứng dụng: tìm việc làm, đăng nhập, đăng ký, thông tin nhà tuyển dụng,…
* **home**: trang chủ, cập nhật các thông báo tuyển dụng mới nhất hoặc được trả phí để đưa lên trang đầu
* **login**: trang đăng nhập dành cho người tìm việc, người đăng tin tuyển dụng, nhà tuyển dụng.
* **qa**: trang hỏi đáp người dùng
* **sign-up**: trang đăng ký người dùng
* **slider**: thành phần hiển thị các tin tuyển dụng
* Component của web được cài đặt có cấu trúc như sau (ví dụ với component header):



* file **.component.js:** chứa mã nguồn javascript điều khiển các hoạt động (behavior) của component
* file **.style.css**: chứa định dạng css cho component
* file **.template.html**: chứa template của component
* Như vậy, để thêm một component trên web ta cài đặt đủ các thành phần file như trên.

## BACKEND

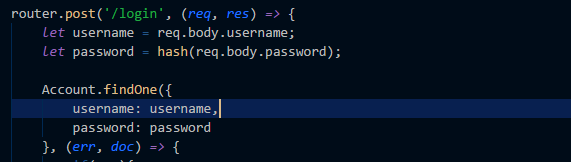
* Thư viện cho server side được lưu trong thư mục **node\_modules** như sau

 … và nhiều thư viện khác.

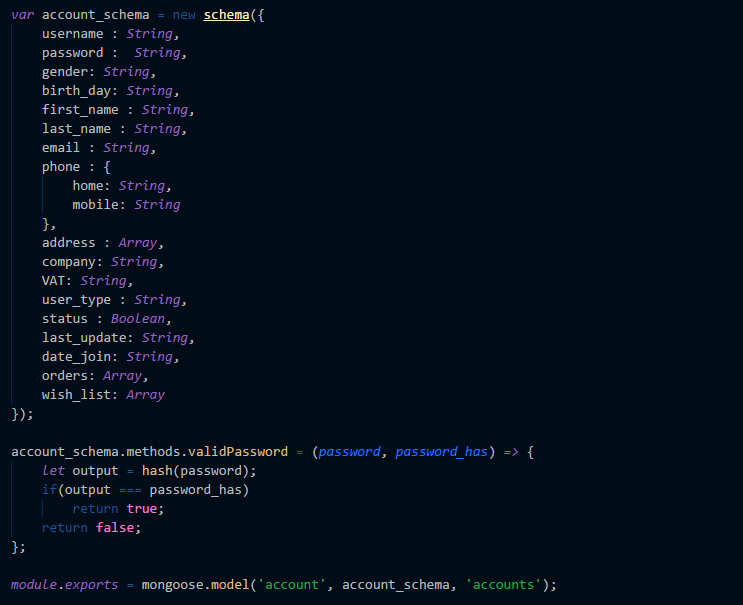
* Sử dụng **express framework** cho việc định tuyến chính các view và api trên webserver và **mongoose framework** cho việc truy xuất dữ liệu trên MongoDB (cơ sở dữ liệu chính của ứng dụng)
* Các component chính trên server side (trong thư mục **server**)



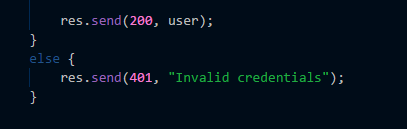
* helpers: chứa các mã nguồn hỗ trợ cho việc xử lý trên server
* middlewares: chứa các mã nguồn cài đặt các middleware (xử lý trung gian) khi định tuyến
* model: chứa các quy định thuộc tính của các đối tượng lưu trong cơ sở dữ liệu
* routes: chứa các mã nguồn cài đặt các api xử lý các định tuyến
* Ứng dụng sử dụng RESTful API, tức chủ yếu truy xuất các kết quả json từ server, ví dụ khi người dùng đăng nhập và thực hiện việc xác thực gửi đến url **/login (trong file routes/api.js)**



* Dữ liệu username và password sẽ được truy xuất và tìm kiểm trong collection chứa thông tin tài khoản thông qua đối tượng **Account**
* Đối tượng **Account** là một model được xây dưng dựa trên một schema quy định thuộc tính của một đối tượng lưu trong collection chứa thông tin tài khoản (**models/account.js**)



* Đối tượng **account-schema** định nghĩa thuộc tính cho một đối tượng lưu trong collection thông tin tài khoản, sau đó xây dựng một model thông qua đó với việc gọi hàm **mongoose.model(‘account’, account\_schema, ‘accounts’).** Kết quả trả về là đối tượng Account trước đó
* Trả về kết quả json cho web client



* **user**: đối tượng json trả về cho client khi xác thực thành công
* **res.send(401,”Invalid credentials”):** xác thực thất bại
* Như vậy, để thêm một api xử lý trong ngữ cảnh RESTful API, ta định nghĩa một xử lý dưới dạng **router./get/post/put/delete/(url, function(req,res))** trong **routes/api.js** và trả về kết quả thông qua đối tượng **res** với lời gọi hàm **res.send(/mã response/,/đối tượng json trả về/)**. Và để định nghĩa thêm một đối tượng trong collection ta định nghĩa qua đối tượng **Schema** của **mongoose** và lấy model thông qua lời gọi hàm **mongoose.model()** – việc cấu hình này phải được lưu trong thư mục **models**