**Tài liệu bàn giao công việc**

*Người bàn giao: Nguyễn Ngọc Hùng*

1. **Dự án Ceres**
2. Ceres Web API(Web Backend):

* Github: <http://13.21.34.44:10000/ceres/ceres_dashboard_backend>

Nhánh phát triển chính: develop

* Clone: git clone http://13.21.34.44:10000/ceres/ceres\_dashboard\_backend.git
* Run: (Lưu ý: Để chạy được hệ thống Ceres Web API cần phụ thuộc vào gói Jar: ceres-common)

B1. Clone **ceres-common**: git clone <http://13.21.34.44:10000/ceres/ceres-common.git>

B2. Checkout sang develop: git checkout develop

B3. Build **ceres-common**: mvn clean install

B4. Clone ceres-dashboard-backend:

git clone <http://13.21.34.44:10000/ceres/ceres_dashboard_backend.git>

B5. Checkout sang develop: git checkout develop

B6. Add external jar: ceres-common-1.0.jar

mvn install:install-file "-DgroupId=com.goldengate.ceres" "-DartifactId=ceres-common" "-Dpackaging=jar" "-Dfile={đường dẫn tuyệt đối}/ceres-common-1.0.jar" "-Dversion=1.0"

(Lưu ý: Cần thay đường dẫn tuyệt đối đến file: ceres-common-1.0.jar)

B7: Để chạy ứng dụng Ceres Web API, chạy lệnh: mvn spring-boot:run

(Có thể dùng IDE để chạy debug).

* Build: mvn package -DskipTests
* **Deploy:**

B1. Sau khi build, lấy file: ceres-web-api.war.original đẩy lên Tomcat trên server

B2. Xoá đuôi .original

B3. Start tomcat

1. Ceres Dashboard(Web Frontend)

* Github: <http://13.21.34.44:10000/ceres/ceres_dashboard>

Nhánh phát triển chính: develop

* Clone: git clone <http://13.21.34.44:10000/ceres/ceres_dashboard.git>
* Run:

B1. Cài thư viện NodeJS(version 14.6.0)

B2. Chạy lệnh: npm install hoặc yarn install

B3. Chạy lệnh: ng serve

* Build: ng build –prod –base-href ./
* Deploy:

Sau khi build thành công, thư mục đã build trong /dist -> Đẩy thư mục này lên server

(Lưu ý: Có thể chỉnh sửa đường dẫn sau khi deploy bằng cách sửa file cấu hình trong:

src/assets/config/config.production.json)

1. **Dự án DeepSignal**

Các module, công nghệ quan trọng cần bàn giao:

+ Notification: Module thông báo dành cho ứng dụng DeepSignal.

+ RabbitMQ: <https://www.notion.so/deepsignal/RabbitMQ-Spring-Boot-5e9c2feeae2b4b60b58fd9b7f838103c>

1. Notification

* Các bảng liên quan:

+ *notification\_type*: Lưu trữ các loại thông báo. Có chức năng định nghĩa template cho các loại thông báo -> cung cấp template cho bảng notification.

Giải thích các trường:

* type: loại thông báo
* template\_content: mẫu nội dung thông báo, có thể truyền data động vào content, được đánh số thứ tự từ 0,1,2…
* template\_title: mẫu tiêu đề thông báo
* template\_url: url khi bấm vào thông báo sẽ được điều hướng tới đây
* name: tên loại thông báo. Khi phát triển, cần thêm mới tên cho thông báo và thêm vào Enum trong code
* category: nếu là interpolation, content có thể truyền nội dung từ ngoài vào. Ngược lại nếu là basic thì content được giữ nguyên từ cột template\_content
* template\_btn: nút của thông báo, có thể truyền html vào cột này
* template\_icon: mẫu icon của từng loại thông báo
* template\_content\_i18n: lưu trữ key đến file i18n của nội dung thông báo. Lưu ý, nội dung trong file i18n phải khớp với mẫu ở cột “*template\_content*”. Nếu cột này rỗng, content sẽ được lấy từ cột “*template\_content*”
* template\_title\_i18n: lưu trữ key đến file i18n của tiêu đề thông báo.Nội dung cột này trong file i18n cũng phải khớp với cột “title”.

+ *notification*: bảng lưu trữ tất cả các thông báo

Giải thích các trường:

* title: Lưu trữ tiêu đề của thông báo
* content: Lưu trữ nội dung của thông báo
* created\_date: thời gian tạo thông báo này
* user\_id: người gửi thông báo này
* type\_id: lấy từ id của bảng “*notification\_type*”
* is\_marked\_read: đánh dấu xem thông báo đã được xem hay chưa.

0 – chưa xem, 1 – đã xem

* username: tên đăng nhập của người gửi thông báo
* is\_checked: kiểm tra xem thông báo đã được check hay chưa(dấu chấm xanh nhỏ cạnh icon của từng thông báo)

0 – chưa kiểm tra, 1- đã kiểm tra

* btn\_content: nút hiển thị trên từng thông báo, lấy từ cột “*btn\_content*” của bảng “*notification\_type*”
* icon\_content: icon của từng thông báo, lấy từ cột “*icon\_content*” của bảng “*notification\_type*”
* type: loại thông báo, lấy từ cột “*type*” của bảng “*notification\_type*”
* content\_i18n: lấy từ cột “*content\_i18n*” của bảng “*notification\_type*”
* title\_i18n: lấy từ cột “*title\_i18n*” của bảng “*notification\_type*”

+ notification\_receiver: Lưu id người nhận của thông báo

* notification\_id: id của thông báo, map với cột id của bảng “notification”
* receiver\_id: id của người nhận, map với cột id của bảng “user”
* Một thông báo có thể có nhiều người nhận

1. RabbitMQ

* Xem chi tiết cách tích hợp RabbitMQ vào dự án SpringBoot và VueJS trên:

<https://www.notion.so/deepsignal/RabbitMQ-Spring-Boot-5e9c2feeae2b4b60b58fd9b7f838103c>

* Giao diện quản lý RabbitMQ:

<http://192.168.10.12:15672/#/queues>

username: deepsignal, password: deeppass22

* Để thêm cấu hình cho RabbitMQ, thêm tại file cấu hình với key:

application.rabbit-config…

* Để thêm một Consumer, có thể thêm một method tại class: com.saltlux.deepsignal.web.api.consumer.RabbitmqConsumer
* Để thêm một Publisher, có thể cấu hình tại: com.saltlux.deepsignal.web.config.RabbitmqConfiguration

1. Websocket

* Backend và frontend cùng phải đăng ký đến 1 điểm cuối(endpoint): /websocket/tracker
* Tất cả các đường dẫn đăng kí đến Stomp, cần phải bắt đầu bằng: /topic.

VD: /topic/notification, /topic/updateFeed

* Tất cả các đường dẫn đăng kí mới đến Websocket đều nằm trong thư mục:

app/service/websocket.service.ts

* Sơ đồ tuần tự cho Websocket:

