

# Dossier de Spécifications Techniques

**IUT Sophia Antipolis**  
**RFID-Web Service**

Gestion du présentéisme en cours

---

Version 2.1.0 du Jeudi 12 Mai 2016

État : Finit

Equipe projet : CARRIE Stéphanie, HELIE-ZADEH Marina, PHILY  
TOM

## Destinataire(s)

IUT Sophia Antipolis  
Etudiant LP SIL IDSE

Cours RFID / Service Web

## Historique

Version	Date	Origine de la mise à jour	Rédigée par	Validée par
1.00	10/01/2016	Création du document	Équipe projet	Équipe projet
1.01	09/01/2016	Ajout du diagramme de séquence	Équipe projet	Équipe projet
1.1	10/02/2016	Ajout des obligations spécifique	Équipe projet	Équipe projet
1.1.1	20/02/2016	Ajout de la structure de la base de données.	Équipe projet	Équipe projet
1.1.2	05/03/2016	Ajout de la présentation des fonctionnalités de L'étudiant	Équipe projet	Équipe projet
1.2	15/03/2016	Ajout des aspects technique de la récupération des données par le lecteur NFC.	Équipe projet	Équipe projet
1.3	01/04/2016	Ajout des propriétés REST	Équipe projet	Équipe projet
2.0	11/05/2016	Ajout des informations d'utilisations et remise en page des spécifications techniques	Équipe projet	Équipe projet
2.1	12/05/2016	Ajout de la partie déploiement du projet	Équipe projet	Équipe projet

# Sommaire

1.	Introduction	5
2.	Glossaire	5
3.	Présentation du projet	5
4.	Structuration des cas d'utilisations	6
4.1.	Présentation des acteurs	6
4.2.	Diagramme de séquence	7
4.3.	Exigences fonctionnelles du cas d'utilisation	8
5.	Base de données	9
5.1.	Modèle relationnel de donnée	9
5.2.	Explication base de données	10
5.2.1.	Table élève	10
5.2.2.	Table cours	10
5.2.3.	Table présence	11
6.	Service REST	12
6.1.	Scan de la carte étudiante	12
6.2.	Les routes administratives	12
6.2.1.	Route connexion de l'administration	12
6.2.2.	Route des cours	12
6.2.3.	Route élève	12
6.2.4.	Route présence	12
6.2.5.	Route insertion Elève	13
6.2.6.	Route insertion Cours	13
6.2.7.	Route insertion feuille présence	13
7.	Serveur Web : NodeJs	14
7.1.	Partie Connexion	14
7.2.	Partie Elève	15
7.2.1.	Avant Scan	15
7.2.2.	Après Scan	15
7.3.	Partie administrateur	16

7.3.1.	Panel administrateur principal	16
7.3.2.	Gestion des étudiants	17
7.3.3.	Gestion des cours	18
8.	Déploiement du projet	19
8.1.	Déploiement de la base de données	19
8.1.1.	Prérequis	19
8.1.2.	Exécution du script DataBuild.sh	20
8.2.	Lancement du server Jetty sous eclipse	20
8.2.1.	Configuration d'eclipse	20
8.2.2.	Configuration de Maven et lancement du serveur	20
8.3.	Configuration et lancement du server NodeJS	21
8.3.1.	Installation des modules NodeJS	21
8.3.2.	Lancement du server	21

## 1. Introduction

Ce document a pour objectif de présenter et de décrire fonctionnellement les cas d'utilisation relatifs au projet de gestion du présentisme en cours dans le cadre du cours de RFID et de web service.

La mise en place de cette application couvre deux axes majeurs :

- La mise en place d'une application faisant appel aux technologies de RFID
- La mise en place d'une base de données pouvant enregistrer des données issues d'une carte sans contact
- La mise en place de web service

## 2. Glossaire

Le tableau ci-dessous donne la liste des acronymes utilisés dans ce document et leur signification :

Acronyme	Signification
RFID	radio frequency identification
NFC	Near Field Connection
IDC	ID de la carte étudiante

## 3. Présentation du projet

Aujourd'hui, l'IUT utilise les feuilles d'émargement sous format papier afin de marquer la présence et l'absence des étudiants. En effet, à chaque début de cours, le professeur fait passer cette feuille afin que chaque étudiant puisse la signer et acter de leur présence. Par la suite ses présences et absences sont enregistré par une personne afin que ces données soient informatisées. Cependant, il arrive que cette feuille d'émargement soit égarée ou abimée, l'enregistrement devient alors plus difficile. Les objectifs de cette application de badgeuse de cours sont de supprimer la double saisie, pallier aux faiblesses de la feuille papier en rendant la saisie du présentisme et de l'absentéisme informatisé dès le début.

La badgeuse serait placée devant d'une salle de cour et serait relié à un écran afin de pouvoir transmettre des informations à l'étudiant. L'écran permettra à l'étudiant de choisir s'il rentre où s'il sort de cours. Une fois ce choix fait, l'étudiant devra badger sa carte étudiante afin de permettre au

système d'enregistrer le numéro de la carte étudiante, le numéro de l'étudiant, l'heure d'arrivée. C'est cet enregistrement qui permettra de mettre en avant le présentéisme, l'absentéisme ou les retards.

Par la suite, l'administrateur devra pouvoir tirer la liste des étudiants par classe et par cours qui ont été présent, absent, retardataire. Pour cela, l'application définira un statut de l'étudiant en comparant son heure d'entrée et son heure de sortie, aux heures des cours de son emploi du temps. L'administrateur pourra faire ses listes en filtrant sur la classe, sur le cours, ou sur les statuts des étudiant (présent, absent, retardataire).

L'application devra pouvoir être capable de lire le numéro de la carte étudiante à l'aide d'un lecteur NFC, d'enregistrer les données qui y sont rattachés, d'enregistrer l'heure lors d'un passage. Par la suite elle devra être capable de définir si un étudiant est présente, retardataire ou absent par rapport à un cours et une plage de donnée définit. Elle devra pouvoir avoir une partie badgeuse et une partie administrative.

## 4. Structuration des cas d'utilisations

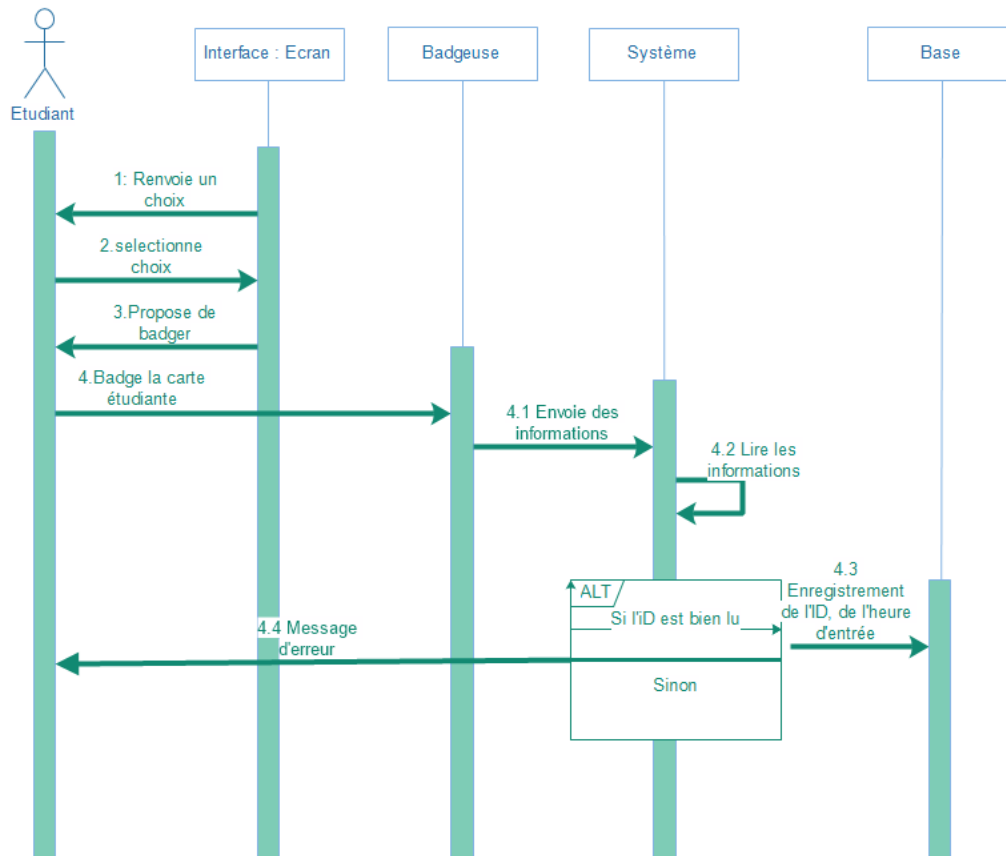
---

### 4.1. Présentation des acteurs

L'application de badgeuse de cours nécessite l'implication de plusieurs acteurs qui interagissent avec le système :

- Acteurs :
  - Etudiant
  - Badgeuse
  - Administrateur
- Condition requise :
  - L'étudiant doit être enregistré dans le système
  - L'étudiant doit posséder une carte étudiante.
  - Les cours doivent être enregistrés en base de données

## 4.2. Diagramme de séquence



### 4.3. Exigences fonctionnelles du cas d'utilisation

La liste des exigences fonctionnelles suivantes devra être couvertes par les cas d'utilisations

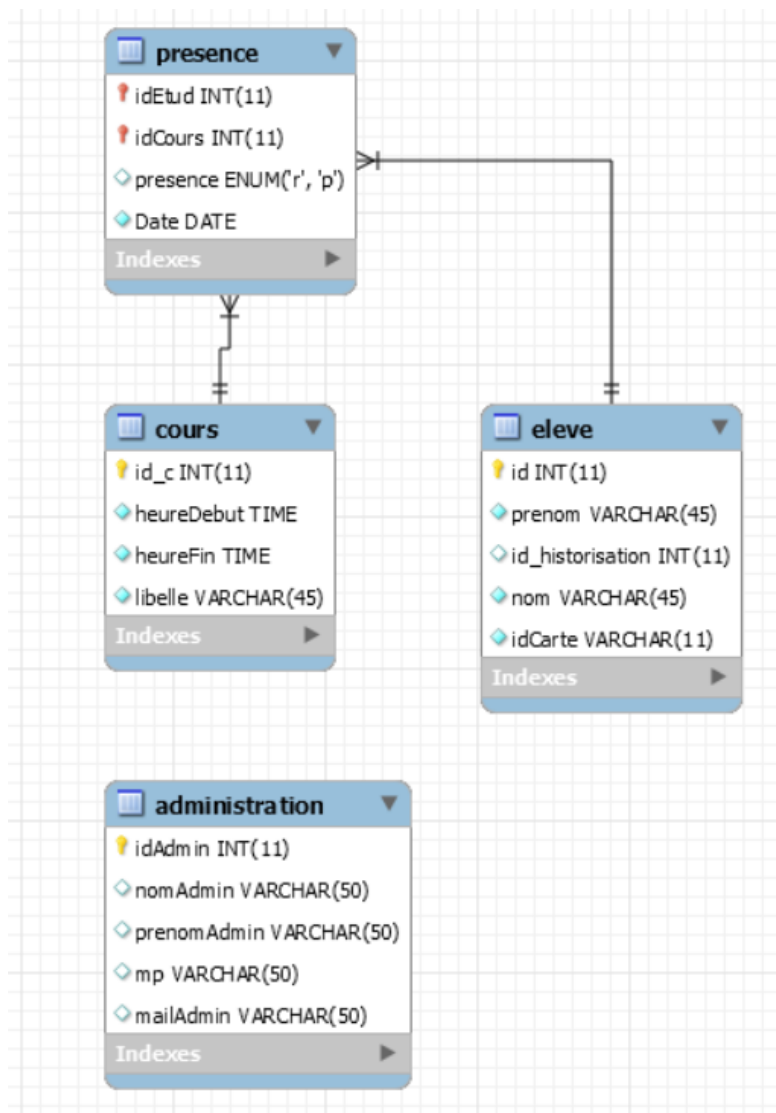
Description générale	
Connexion Etudiant	L'utilisateur de type étudiant devra pouvoir se connecter à l'application afin de pouvoir badger sa carte par la suite.
Badge étudiant	L'étudiant devra pouvoir badger sa carte étudiante afin de pouvoir enregistrer son entrée et sa sortie de cours.
Enregistrement	Le système devra pouvoir prendre en compte l'enregistrement du badge de la carte étudiante afin d'enregistrer en base de données la présence de l'étudiant en cours.
Connexion administrateur	L'administrateur devra pouvoir se connecter à l'application afin de réaliser diverses actions d'administration.
Regarder la feuille de présence	L'administrateur devra pouvoir regarder la liste des élèves absent et présent à un cours donnée
Générer une feuille de présence	L'administrateur devra pouvoir générer une feuille de présence pour un cours donnée.
Gérer la liste d'élève	L'administrateur devra pouvoir ajouter, supprimer ou modifier un élève
Gérer la liste des cours	L'administrateur devra pouvoir ajouter, supprimer ou modifier un cours.



## 5. Base de données

Afin de bien fonctionner, l'application possède une base de données sous Mysql Worbench 6.3.

### 5.1. Modèle relationnel de donnée



## 5.2. Explication base de données

La base de données porte le nom de rfid\_badgeuse, elle comporte quatre tables.

### 5.2.1. Table élève

La table élève permet d'enregistrer l'ensemble des élèves. Elle devra au préalable être remplie avec l'ensemble des élèves afin de leur permettre de badger leur carte étudiante pour attester de leur présence. La table est composée de cinq champs :

- ID : Ce champ permet d'identifier un élève dans la base de données. L'Id étudiant correspond à un seul et unique étudiant. Ce champ sera la clé primaire de cette table. Il prendra le format Integer(11)
- PRENOM : Ce champ correspond au nom de l'étudiant. Il prendra le format Varchar(45).
- NOM : Ce champ correspond au prénom de l'étudiant. Il prendra le format Varchar(45).
- IDCARTE : Ce champ correspond au numéro de la carte étudiante d'un élève. C'est ce numéro qui est récupéré une fois la carte badger. Ce champ est au format Varchar(11).
- ID\_HISTORISATION : Ce champ correspond à l'historisation de la carte étudiante. En effet, dans le cas de la perte/vol d'une carte, ce champ permet d'historier les cartes précédentes de l'étudiant. Il prendra le format Varchar(11).

### 5.2.2. Table cours

La table cours permet d'enregistrer l'ensemble des cours pour une heure donnée. Elle devra être au préalable remplie avec une liste de cours disponible au quel un étudiant peut participer. La table est composée de quatre champs :

- ID : Ce champ permet d'identifier un cours dans la base de données. L'Id cours correspond à un seul et unique cours. Ce champ sera la clé primaire de cette table. Il prendra le format Integer(11).
- HEUREDEBUT : Ce champ correspond à l'heure de début du cours. Il ne peut pas être nul. Il prendra le format TIME.
- HEUREFIN : Ce champ correspond à l'heure de fin du cours. Il ne peut pas être nul. Il prendra le format TIME.
- LIBELLE : Ce champ correspond au libelle du cours. C'est ce champ qui permet de désigner le cours. Il prendra le format Varchar(45).

### 5.2.3. Table présence

La table présence correspond à la table qui enregistra la présence des étudiants à un cours donnée. Elle devra être prés rempli avec la liste des étudiants qui devront assister à un cours précisé. Cette table sera composée de quatre champs :

- IDETUDIANT : Ce champ correspond à l'ID de l'étudiant présent dans la table élève qui permet d'identifier un élève à un cours. Il compose la clé primaire composite de cette table. Il prendra le format Integer(11)
- IDCOURS : Ce champ correspond à l'ID du cours présent dans la table cours qui permet d'identifier le cours aux quel la liste d'élève doit participer. Il prendra le format Integer(11)
- PRESENCE : Ce champ correspond au marqueur de présence de l'étudiant. Il prendra le format Enum avec les valeur P pour présent et A pour absent. Au début du cours, ce champ est nul pour l'ensemble des étudiants.
- DATE : Ce champ correspond à la date à laquelle l'étudiant badge sa carte. Il prendra le format DATE.

## 6. Service REST

---

### 6.1. Scan de la carte étudiante

Lorsque l'étudiant demande à pouvoir scanner sa carte, le serveur NodeJs transmet la demande au service REST afin que celui-ci prenne en charge la lecture. Une fois, que le service REST a effectué la lecture de la carte étudiante, celui-ci récupère le numéro de la carte qui a été scannée. Grâce à celui-ci, le service REST récupère l'ensemble des données de l'étudiant en base de données afin de mettre à jour le champ présence de la table présence avec la valeur 'P'. Enfin, il renvoie l'ensemble des informations de l'étudiant au serveur web sous NodeJS.

### 6.2. Les routes administratives

La partie administration de l'application sera composée de sept routes définies de la manière suivante.

#### 6.2.1. Route connexion de l'administration

L'utilisateur d'administration accède à la page de connexion grâce au serveur Web. Après la saisie des données de connexion, le serveur NodeJs envoie les données au service REST, qui vérifie en base de données les informations de connexions. Si celle-ci sont justes, l'utilisateur est renvoyé vers la route administrateur du serveur NodeJs, sinon il reste sur la route de connexion admin avec un message d'erreur.

#### 6.2.2. Route des cours

Cette route permet à l'administrateur de récupérer la liste de l'ensemble des cours présents en base de données. Le serveur NodeJs renvoie la demande au service REST qui lui remonte l'ensemble des données présentes en base.

#### 6.2.3. Route élève

Cette route permet à l'administrateur de récupérer la liste de l'ensemble des élèves présents en base de données. Le serveur NodeJs renvoie la demande au service REST qui lui remonte l'ensemble des données présentes en base.

#### 6.2.4. Route présence

Suite au choix de l'administrateur d'un cours et d'un jour, celui-ci peut récupérer la liste de présence des élèves à ce cours. La demande est transmise au service REST via le client web NodeJS. C'est le service REST qui renverra alors les données de la base de données.

#### 6.2.5. Route insertion Elève

L'administrateur aura la possibilité de rentrer les informations de l'élève. Ainsi il rentre dans un premier temps les données standard de l'étudiant. Ces données sont transmises au service REST qui demande alors le scan de la carte étudiante afin de remplir les informations. Une fois l'ensemble des données collectés le service REST effectue l'insertion en base de données.

#### 6.2.6. Route insertion Cours

L'administrateur aura la possibilité de rentrer les informations d'un cours. Une fois fait, l'ensemble des données sont transmises au service REST via le service web qui insert par la suite les données en base.

#### 6.2.7. Route insertion feuille présence

L'administrateur aura la possibilité de rentrer les informations de la feuille de présence. Une fois fait, l'ensemble des données sont transmises au service REST via le service web qui insert par la suite les données en base.

## 7. Serveur Web : NodeJs

---

Le serveur Web NodeJs permet aux utilisateurs d'utiliser l'application de badgeuse. L'IHM est composé en trois grosses parties.


### 7.1. Partie Connexion


Lorsque le server NodeJS est lance, l'utilisateur arrive sur une page de connexion correspondant à la route NodeJs . Il doit alors choisir La catégorie que le concerne : étudiant ou administrateur.



# Badgeuse Etudiante

**Connectez-vous**

 Partie Etudiant

 Partie enseignant

## 7.2. Partie Elève

Lorsque l'utilisateur sélectionne la partie étudiante, celui-ci le renvoie vers une page qui lui propose de scanner sa carte. Il faut au préalable que l'utilisateur appuie sur le bouton scanner ma carte.

### 7.2.1. Avant Scan

Une fois le choix fait de se connecter par l'étudiant, celui-ci est renvoyé vers une page lui proposant de scanner sa carte qui correspond à la route NodeJs « /badgeetudiant ». Afin de pouvoir scanner sa carte il est nécessaire de cliquer au préalable sur le bouton lecture. Lorsque le bouton activé NodeJs envoie la demande au service REST pour la lecture de la carte.



## Badgeuse Etudiante

**Badgeuse Etudiante**

Scanner votre carte

lecture

← Retour

### 7.2.2. Après Scan

Une fois la carte scannée et interprétée par le service REST celui-ci renvoie alors un message positif si l'enregistrement de la présence de l'étudiant c'est bien passé, soit un message d'erreur dans le cas contraire. C'est le service REST qui gère la mise à jour de la présence de l'étudiant en base de données.

### 7.3. Partie administrateur

Lorsque le serveur NodeJs est lancé et que l'utilisateur se retrouve sur la page de connexion, l'administrateur a le choix de pouvoir se connecter à son espace dédié. Il se retrouve alors sur la route NodeJs « /logprof » proposant une page de connexion standard.



## Badgeuse Etudiante

Connectez-vous

Votre Email

Mot de passe

Connexion

Retour

l'ensemble des route administrateur qui celui-ci soit bien connecté.

Pour se connecter à la partie administrateur utiliser les données de connexion suivante : login : [admin.badgeuse@unice.fr](mailto:admin.badgeuse@unice.fr) , mot de passe : admin

Afin d'assurer également la sécurité de l'application, il n'est pas possible de rajouter un administrateur via L'IHM, de plus pour une question de confidentialité, le mot de passe est encrypté en SHA512 dans la base de données.

#### 7.3.1. Panel administrateur principal

Une fois connecté et identifié l'administrateur est redirigé vers le panel d'administration à savoir la route « /admin » de NodeJs. Trois choix est alors possible :

- Gestion des étudiants : cette partie permet à l'administrateur de gérer l'ensemble des données des étudiants.
- Gestion des cours : cette partie permet à l'administrateur de gérer l'ensemble des données relatives aux cours, ainsi que de générer la feuille de présence.
- Déconnexion : permet à l'administrateur de quitter la partie d'administration.



## Badgeuse Etudiante

Administration

Gestion des Etudiants

Gestion Cours

Déconnexion



### 7.3.2. Gestion des étudiants

Lorsque l'administrateur accède à la gestion des étudiants, il est alors redirigé vers la route « /gestionE » de NodeJs. Il lui sera possible alors d'effectuer les quatre actions suivantes :

- Liste des étudiants : Ce bouton redirige vers la route NodeJs « /listeE », lui permettant de visualiser l'ensemble des étudiants inscrits dans la base de données. Il peut ainsi voir les données suivantes :
  - Le nom
  - Le prénom
  - Le numéro de la carte étudiante
  - Le numéro de la carte étudiante historiques.
- Ajouter étudiant : Ce bouton redirige vers la route NodeJs « /ajoutE », elle permet d'ajouter un étudiant en base de données. Il faudra alors à l'administrateur renseigné :
  - Le nom de l'étudiant
  - Le prénom de l'étudiant
  - Scanner la carte de l'étudiant afin de pouvoir enregistrer le numéro de la carte
- Modifier étudiant : Ce bouton redirige vers la route NodeJs « /modifE », elle permet de modifier les données d'un étudiant à savoir :
  - Le nom de l'étudiant
  - Le prénom de l'étudiant
  - Modifier le numéro de la carte étudiante en scannant une nouvelle carte. Dans ce cas-là, la carte précédente sera historisée en base de données.
- Supprimer un étudiant : Ce bouton redirige vers la route NodeJs « /suppE », cette route permet à l'administrateur après avoir sélectionné l'étudiant peut le supprimer.



## Badgeuse Etudiante



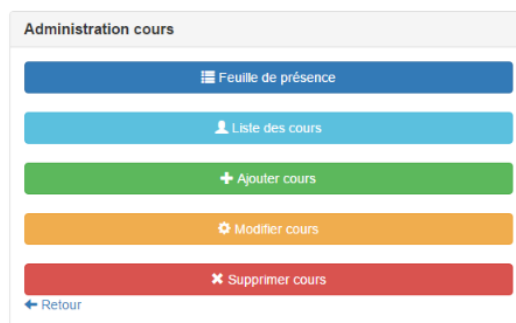
### 7.3.3. Gestion des cours

Lorsque l'administrateur accède à la gestion des cours, il est alors redirigé vers la route NodeJs « /gestionC ». Il lui sera alors possible d'effectuer les cinq actions suivantes :

- Feuille de présence : Ce bouton redirige vers la route NodeJs « /presenceC ». Ainsi, l'administrateur peut sélectionner un cours et une date afin de récupérer la liste des présences à ce cours. Il récupérera les données des étudiants absent et présent.
- Liste des cours : Ce bouton redirige vers la route NodeJs « /listeC », lui permettant de visualiser l'ensemble des cours dans la base de données. Il peut ainsi voir les données suivantes :
  - Nom de la matière
  - Heure de début du cours
  - Heure de fin du cours
- Ajouter un cours : Ce bouton redirige vers la route NodeJs « /ajoutC », elle permet d'ajouter un cours en base de données. Il faudra alors à l'administrateur renseigné :
  - Nom de la matière
  - Heure de début du cours
  - Heure de fin du cours
- Modifier un cours : : Ce bouton redirige vers la route NodeJs « /modifC », elle permet de modifier les données d'un cours à savoir :
  - Nom de la matière
  - Heure de début du cours
  - Heure de fin du cours
- Supprimer un étudiant : Ce bouton redirige vers la route NodeJs « /suppC », cette route permet à l'administrateur après avoir sélectionné le cours, la date, il peut le supprimer.



## Badgeuse Etudiante



## 8. Déploiement du projet

---

L'accès au projet se fait via GitHub à l'adresse suivante : <https://github.com/PhilyT/nfcProject>

Le projet de la badgeuse étudiante est composé des fichiers suivant :

- ClientNodeJs
  - Css
  - Fonts
  - Img
  - Public
  - View
  - Package.json
  - Server.js
- Doc
- src/main
  - Java
  - Webapp
- .classpath
- .gitignore
- BuildDatabase.sql
- DataBuild.sh
- README.md
- Pom.xml

Avant le déploiement du projet vérifier la bonne structure du projet comme présenter ci-dessus. Le déploiement du projet se fait en trois étapes :

- Déploiement de la base de données
- Déploiement du serveur Jetty sous eclipse
- Lancement du serveur nodeJs

### 8.1. Déploiement de la base de données

Afin de configurer la base de données, deux fichiers sont nécessaires : DataBuild.sh et BuildDatabase.sql

#### 8.1.1. Prérequis

Pour pouvoir exécuter le script sh de déploiement vérifier la possibilité d'exécuter les commandes de mysql dans une invite de commande standard. Si une erreur est retournée, vérifier les variables d'environnement afin de rajouter celle de Mysql.

### 8.1.2. Exécution du script DataBuild.sh

Dans git, taper la commande suivante : sh DataBuild.sh. Les données suivantes sont nécessaires :

- Hôte : localhost
- User : root
- Mot de passe : mot de passe de votre user root

Le script vérifie dans un premier temps si la connexion à mysql est possible. Puis si la base existe déjà, si c'est le cas il la supprime. Dans un troisième temps le script exécute le fichier BuildDatabase.sql qui crée la base, insère un minimum de données et crée l'utilisateur rfid qui permettra la communication avec la base de données par le service REST (login : rfid, mot de passe : rfid). Enfin dans un dernier temps, le script vérifie la bonne existence de la base de données.

## 8.2. Lancement du serveur Jetty sous Eclipse

Afin de déployer le projet, certaines configurations sont à effectuer sur Eclipse :

- Configuration d'Eclipse
- Configuration de Maven

### 8.2.1. Configuration d'Eclipse

Configuration des préférences Java afin de remplacer le JRE par le JDK. Pour cela aller dans :

→ Window → Préférences → Java → JRE installer → edit jre par défaut → directory → sélectionner le JDK à la place du JRE → appliquer.

### 8.2.2. Configuration de Maven et lancement du serveur

Configuration de Maven :

→ Clic droit sur le projet → run as → run configuration → nouvelle configuration maven → Mettre dans goal : clean install jetty :run → mettre sur la même page dans base directory le répertoire du projet.

Lancement du serveur :

→ Clic droit sur le projet → run as → run configuration → Aller dans la configuration Maven créée précédemment  
→ Clic sur Run

### 8.3. Configuration et lancement du server NodeJS

Afin de lancer le server NodeJs deux actions sont à réaliser :

- Installation des module NodeJs
- Lancement du server

#### 8.3.1. Installation des modules NodeJS

Ouvrir une invite de commande dans le dossier ClientNodeJs. Taper la commande suivante :

- npm install

L'installation peut prendre jusqu'à deux minutes.

#### 8.3.2. Lancement du server

Dans la même invite de commande que précédemment, taper la commande suivante afin de lancer le server NodeJs :

- node server.js