

Rapport projet MPA

Emarge tes élevés !!

Chiara RELEVAT – Alan VADELEAU – Yann CHANTREL

Univesite de bretagne occidental

novembre 2020 – mars 2021

# Introduction

Notre projet s’intitule Émarge tes élèves !!, il a pour but d’implémenter une application web et mobile, avec la technologie React-Native. Cette application permet aux enseignants de renseigner la présence des élèves lors de leurs cours.

Il a été réalisé en équipe de trois, par Alan VADELEAU, Chiara RELEVAT, et Yann CHANTREL, en collaboration avec Madame Laurence DUVAL, notre enseignant référent.

La première étape pour la réalisation du projet a consisté à apprendre les spécificités du langage React-Native qui nous était alors inconnu. Pour cela, notre enseignant référent nous a communiqué un tutoriel sur internet, à partir duquel nous avons réalisé une petite application.

Nous avons ensuite établi, ensemble, le cahier des charges. Puis nous avons effectué toute la conception, pour l’IHM, et pour la base de données. Ensuite nous avons implémenté le code nécessaire. Le tout en étroite relation avec notre enseignant référent, que ce soit pour vérifier que nous nous dirigions vers la bonne voie, ou pour répondre à nos questions d’ordre technique.

Enfin nous avons présenté notre application, en janvier. Celle-ci comportait encore des bugs. Nous avons profité des dernières semaines pour les corriger, et implémenter une connexion CAS, pour que les enseignants puissent se connecter directement avec leurs identifiants ENT.

Ce document va vous permettre de prendre connaissance du projet. Vous trouverez également toutes les explications nécessaires pour utiliser et reprendre ce projet.

Table des matières

[Introduction 1](#_Toc66460650)

[1. Cahier des charges 3](#_Toc66460651)

[2. Conception 4](#_Toc66460652)

[2.1. IHM 4](#_Toc66460653)

[2.2. Base de données 4](#_Toc66460654)

[3. Mise en place du projet 5](#_Toc66460655)

[4. Implémentation du projet 6](#_Toc66460656)

[4.1. Côté serveur 6](#_Toc66460657)

[4.2. Côté client 6](#_Toc66460658)

[4.2.1. Changement de page 6](#_Toc66460659)

[4.2.2. Gestion des onglets 6](#_Toc66460660)

[4.2.3. Liste et Pop-up 6](#_Toc66460661)

[5. Ressources pour la reprise du projet 7](#_Toc66460662)

[6. Conclusion 8](#_Toc66460663)

[7. Annexe 9](#_Toc66460664)

[7.1. Manuel d’installation 9](#_Toc66460665)

[7.2. Manuel d’utilisation 9](#_Toc66460666)

# Cahier des charges

Nous avons établi un cahier des charges avec notre enseignant référent en début de projet. Il a ensuite été légèrement modifié lors de notre deuxième période pour ce projet, c'est-à-dire après notre présentation, en janvier. Notre cahier des charges final comporte les objectifs suivants :

* Un enseignant doit pouvoir se connecter sur l’application via le CAS.
* Un enseignant doit pouvoir se déconnecter à tout moment.
* Pour la création d’une séance :
  + L’enseignant doit pouvoir renseigner une UE, un type de cours (cours, TD, ou TP), un groupe, une date, et un créneau horaire de début et de fin.
  + Le champs UE et groupe doivent proposer une liste de choix restreinte pour chaque enseignant. Un enseignant doit seulement pouvoir choisir des éléments dont il fait partie, ou qu’il gère.
  + Le créneau de début doit avoir par défaut l’heure actuelle, le créneau de fin doit avoir par défaut l'heure actuelle + 2 heures. Enfin l’heure de fin ne peut être inférieure à l’heure de début.
  + Il doit être possible pour l’enseignant de créer une nouvelle UE, celle-ci devra être visible dans sa liste des UE après l’ajout.
  + Après la création de la séance, la séance qu’il vient d'être créé doit apparaître dans l’historique des séances.
* Pour l’historique :
  + L’historique des séances doit comporter toutes les séances que l’utilisateur a créé, aujourd’hui, comme précédemment.
  + Au clic sur une séance, il doit voir la liste de présence correspondante.
  + Les séances sont rangées par ordre antichronologique par rapport à leur date de création. La séance la plus récemment créée se retrouve donc en haut de la liste.
* Pour la liste d’appel :
  + Pour chaque étudiant de la liste, il doit être possible de cocher ou non sa présence.
  + En cliquant sur l'étudiant, on peut indiquer si celui-ci est présent, non présent sans justification, ou non présent avec justification. Un champ de texte sera prévu pour la justification. On doit aussi voir sa photo.
  + L’enseignant peut ajouter des étudiants dans la liste d'appels.
  + L’enseignant peut supprimer des étudiants dans la liste d'appels.
  + Les étudiants sont rangés par ordre alphabétique des noms.
  + Une fois la validation de l’appel faite, vous êtes redirigé vers l’historique des séances. Il est toujours possible de changer l’état des présences après validation.

# Conception

## IHM

La conception de l’interface a été faite avec balsamique cloud. Notre application devait être utilisable en version web et mobile. Nous avons donc imaginé une interface pour mobile qu’on adaptera en web.

La partie connexion est entièrement gérée par le CAS nous n’avons donc pas à nous en soucier.

Une fois connecté on veut que l’utilisateur puisse rapidement accéder à la création d’une séance ou à l’historique de ses séances. C’est pourquoi nous avons mis des onglets en bas de l'écran.

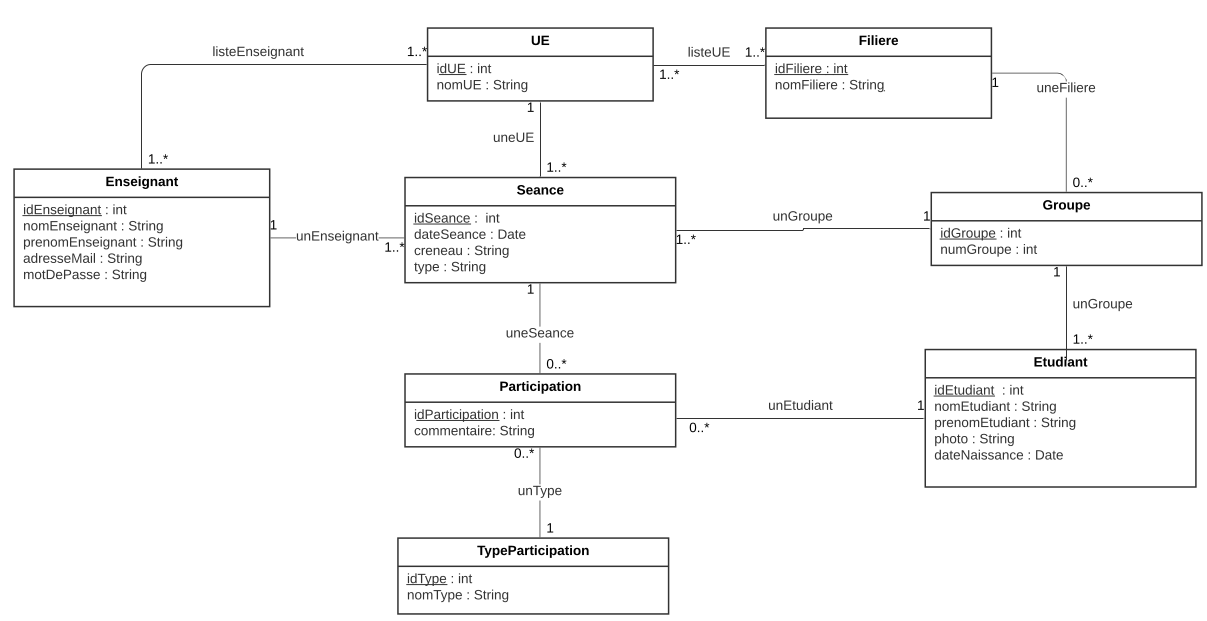
Après sa connexion il sera par défaut sur l’onglet de création d’une séance. Ou il pourra facilement y entrer toutes les informations. L’application ayant identifié l’enseignant, seuls les UE et groupe de l’enseignant seront disponibles. S’il souhaite enseigner un cours dans une UE qu’il n’a pas il peut toujours l’ajouter grâce au bouton plus. Pour lui faciliter la tâche, la date et l’heure actuelle seront saisies par défaut.

Une fois la séance créée, il faut aller dans l’onglet historique qui nous permettra de faire l’appel. A partir de là on peut choisir le type d’absence pour chaque élève. Si on coche juste la case c’est qu’il est présent. Si on ne la coche pas c’est une absence simple. Et si on veut préciser le type d'absence (il peut être intéressant de savoir si elle est justifiée) on peut cliquer sur l’élève et préciser le type d’absence en plus de pouvoir laisser un commentaire et éventuellement voir une photo de l’étudiant.  
Enfin, on peut supprimer un étudiant d’une liste en cliquant sur la poubelle à côté de son nom. Il est également possible de rajouter un étudiant grâce au bouton d’ajout en bas de la liste.

Quand on à créer une ou plusieurs séances on peut les consulter dans l’onglet historique. Il y a une liste des séances triée par ordre antichronologique. On peut alors consulter la liste d’appel de cette séance en cliquant dessus. On pourra également à nouveau changer les absences de chaque élève.

## Base de données

La base de données de l’application va contenir toutes les séances mais également les informations sur les différents cours et enseignement. Elle est organisée de la manière suivante :

Ce qui est important à comprendre c’est comment marche la création d’une séance et ses participations. Dans un premier temps, une séance est propre à un groupe, donc pour chaque étudiant du groupe on va créer une participation liée à cette séance.   
Chaque participation correspond à un type de participation qui peut être une présence, une absence injustifiée ou une absence justifiée.   
L'intérêt d’avoir créé une table pour cela est que seulement ces types d’absence sont valable, aucun autre type ne pourra être créé par l’application en cas d'erreur. Si l’on veut ajouter un type de participation il faut l’ajouter dans cette table.

# Mise en place du projet

# Implémentation du projet

Pour comprendre cette partie il faut avoir des bases en JavaScript et en React-Native.

## Côté serveur

### 4.1.1 Base de données

Toutes nos requêtes à destination de notre base de données passent au préalable sur notre “serveur intermédiaire”, celui-ci redirige les requêtes vers notre “serveur Bdd” qui exécute nos requêtes sur notre base de données MySQL.  
  
Cette configuration permet de simplifier et de diminuer nos requêtes, de nos pages vers le serveur intermédiaire, celui-ci fait ensuite toutes les requêtes basiques ou supplémentaires nécessaires pour réaliser notre tâche.   
  
Cette méthode comporte également l’avantage de ne pas avoir besoin de recréer toutes nos requêtes dans le cas où nous devrions changer notre base de données. Seul notre “serveur Bdd” serait alors à modifier.   
  
On peut également penser au cas où une deuxième application pourrait utiliser la même base de données, cette configuration diminuerait alors, pour nos deux applications, l’implémentation de requêtes non utilisées.

## Côté client

Pour comprendre cette partie il faut connaître quelques principes importants de React-Native :

* Dans une classe il y une fonction render() qui va retourner du code HTML qui sera le rendu sur le navigateur ;
* Dans une classe il y a un constructeur contenant des states qui sont des variables. A chaque changement d’un state la fonction render() est appelée à nouveau ;
* On peut faire passer des variables d’une classe à une autre en les mettant en paramètre de la classe destination. La classe destination pourra y faire appel en utilisant les mot clés this.props.

### Changement de page

Une des particularité du react-native (comme pour le react) c’est qu’il permet de créer des “single page application”, c'est-à-dire des applications qui ne permettent pas le changement de page comme un site web plus classique. Il nous a donc fallu mettre en place un système pour simuler un changement de page.

C’est notre App.js qui permet d’afficher la page que l’on souhaite. Par défaut la page charger est notre Connexion.js, nous changeons le contenu du render() de App.js en fonction d’une variable présente dans les states.

Plus concrètement, dès que l’utilisateur réalise une action qui nécessite un changement de page, nous appelons une fonction parente, changePage(nom), pour changer la valeur du state de App.js. Les states ayant été modifiées, le render() est rappelé, celui-ci affiche la page correspondante en fonction de la valeur du state que nous avons modifié, la nouvelle page est donc affichée.

changeEtat(valeur) est obtenu par les objets enfant de App.js puisque cette fonction leur est transmise à leur création. Elle est ensuite récupérée via this.props.



Figure 1 - State de la classe App.js

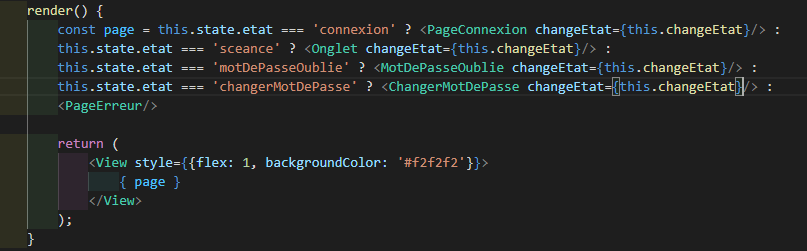


Figure 2 - Render() de la classe App.js

Par exemple ici le state etat à la valeur ‘connexion’ donc la constante page va avoir la valeur <PageConnexion changeEtat={this.changeEtat}/>. Ce sera donc la page de connexion qui sera affiché.

Une fois une classe charger elle pourra changer de page en faisant appel à la fonction changeEtat(valeur) qu’on lui aura transmis en paramètre.



Figure 3 - méthode changeEtat de la classe App.js

Il suffit de lui passer le nom de la page en paramètre pour qu’elle mette à jour le state etat pour effectuer le changement de page.

### Gestion des onglets

### Liste et Pop-up

# Ressources pour la reprise du projet

# Conclusion

# Annexe

## Manuel d’installation

## Manuel d’utilisation