Keep co rapport

Contents

[Introduction 3](#_Toc504552421)

[1- Organisation du projet 4](#_Toc504552422)

[a) Répartition des tâches 4](#_Toc504552423)

[b) Gestion des objectifs 4](#_Toc504552424)

[c) Logiciels de gestion de projet 4](#_Toc504552425)

[1. Slack 4](#_Toc504552426)

[2. Trello 4](#_Toc504552427)

[3. GitHub 4](#_Toc504552428)

# Introduction

Dans le cadre de notre majeure “Informatique du Web et Objets communicants”, nous avons eu l’opportunité d’effectuer un projet de 100 heures regroupant toutes les connaissances que nous avons pu acquérir lors de notre formation.

Partant du constat que, de plus en plus de jeunes partent faire leurs études ou travailler à l’étranger, nous avons eu dans l’idée de créer une application permettant aux familles aux 4 coins du monde de rester en contact, en regroupant plusieurs moyens de communication et partage de médias.

Afin de répondre aux contraintes imposées par le format du projet nous avons divisé notre travail en 4 parties distinctes : une partie front-end mobile, une front-end WEB, une partie back-end dédiée aux données échangées et enfin une partie back-end dédiée aux données utilisateurs.

C’est à travers ce rapport que nous allons vous expliquer l’organisation du projet, les choix fonctionnels et techniques réalisés, les périmètres fonctionnels et enfin les perspectives d’évolution.

# Organisation du projet

## Répartition des tâches

Afin d’optimiser au maximum notre travail et l’avancement de notre projet nous avons découpé notre projet en 4 parties distinctes comme énoncé précédemment. Chaque membre de l’équipe KeepCo était en charge d’une partie développement :

Pia Monge, Développeur Mobile

Julien Vercamer, Développeur WEB

Hugo Subtil, Développeur Back-end (données échangées)

Olivier Thirion, Développeur Back-end (données utilisateurs)

## Gestion des objectifs

Chaque début de semaine étaient décidés les objectifs hebdomadaires, afin de fixer des deadlines au projet et de suivre les avancements et blocages possibles. Cette méthode nous a été inspirée des méthodes de programmation agile, permettant un suivi de projet et une remise en question plus facile.

Chaque fin de journée, un debrief sur les avancés de chacun était effectué, afin de déceler plus facilement et rapidement les difficultés et ainsi renforcer les efforts sur les points bloquants.

## Logiciels de gestion de projet

Afin de poser une communication et une gestion de projet plus facile et fluide nous avons utilisé 3 logiciels de gestion de projet : Slack, Trello et GitHub.

### Slack

Nous avons choisi d’utiliser Slack comme moyen de communication pour notre projet. Grâce à ce logiciel nous avons pu créer plusieurs conversation (une pour chaque partie du projet) afin de ne pas mixer les échanges d’informations et de n’avoir que les informations importantes pour chaque partie. De plus nous avons lié notre logiciel Slack à notre dossier GitHub, contenant notre projet, afin d’avoir accès et notification dès qu’un Commit était effectué.

### Trello

Afin de se fixer des objectifs plus facilement et de donner accès à l’état des avancements de chacun à tous les membres de l’équipe, nous avons divisé chacune de nos parties en « carte » ayant 3 états possible : *A faire, En cours, Terminé*. Dès qu’une tâche/carte, était démarrée elle passait dans en cours, et dès qu’elle était testée et terminée elle passait dans terminé. Ainsi chaque membre de l’équipe pour suivre les avancements de chacun et mesurer la quantité de travail restant pour atteindre les objectifs.

### GitHub

Pour gérer un projet sur lequel 4 personnes allaient effectuer des modifications nous avons décidé d’utiliser le logiciel de contrôle de version GitHub. Chaque partie avait sa branche de développement et lorsqu’une version stable (sans bugs) était finalisée, alors elle était push sur la branche master. Grâce à cette méthode nous gardions de manière constante une version présentable et fonctionnelle à présenter.

# Spécifications fonctionnelles et techniques

## Spécifications front-end WEB

Nous avons choisi React pour plusieurs raison. Premièrement, React apporte une grande rapidité et fluidité dans l’expérience utilisateur grâce à sa gestion du DOM. De plus, l’organisation en composant apporte de la lisibilité et de la ré utilisabilité au code, ce qui facilitera la maintenance et apportera une valeur supplémentaire à la solution.Enfin, React joui d’une grande popularité dans le monde du développement Web actuellement, ce qui a pour avantage de rallier une grande communauté de développeur, et donc de potentiel solution à un problème préci.

Nous avons également utilisé une collection d’élément lié à la navigation appelé react-router ainsi qu’un framework React-bootstrap afin de faciliter le développement. La non utilisation de Redux est dû à une erreur de jugement dès le départ du projet, et également à un manque de connaissance sur ce principe.

### Navigation

Nous avons utilisé react-router pour naviguer entre nos différentes pages. Le principe est le suivant : à chaque URL correspond un composant. La page App.js qui contient la barre de navigation est chargé, avec un composant en plus en fonction de l’URL.Cela est rendu possible par le composant Routes, qui recense l’ensemble des URL associé à un composant en particulié.

Cependant afin de pouvoir communiquer des informations entre ces pages, comme par exemple la socket de communication et diverses variables lié à l’identification et la navigation, il a fallu créer un composant AppliedRoute englobant Route et contenant les informations à transmettre.

Lorsqu’une connexion avec succès d’un utilisateur se produit, diverses informations, telles qu’un token et le rôle ( USER ou ADMIN ) de l’utilisateur, sont stockées en local storage. En fonction du rôle ADMIN ou USER, une redirection s’effectue vers la page consacrée à l’administration ou vers la page d’accueil d’un utilisateur enregistré.

De plus, diverses variables lié à l’authentification sont mise à jour et seront transmissent de page en page lors de la navigation, afin de s’assurer que l’utilisateur soit bien toujours connecté, même en cas de refresh de page ou de navigation via URL (en changeant manuellement l’URL ).

### Communication avec le serveur

La communication avec le serveur se fait via une socket ( socket.io ). Une socket est créé avec la création du composant App et est transmise entre les différentes pages. Une unique socket se charge donc de la totalité des échanges serveur / client spécifique.

Le choix d’une communication par socket est plus explicité dans la partie « Back-end données échangées » de ce rapport.

### Inscription

Une page d’inscription est disponible via la barre de navigation lorsqu’aucun utilisateur n’est connecté au service. Cette page se compose de la barre de navigation ainsi que d’un composant Signup contenu dans le fichier *Signup.js* . Dans ce composant, on retrouve dans la fonction *render()* un formulaire contenant des champs issu de react-bootstrap. A chaque modification d’un de ces champs, un json dans le state est mis à jour, puis envoyé au serveur dès que le bouton *Envoyer* est activé.

Une fonction bloque l’envoie du formulaire tant que des conditions ne sont pas remplies, telle qu’un format d’adresse valide, ou encore lorsque les champs de vérification du mot de passe et de l’adresse mail de ne sont pas identiques au champs associés. L’envoie du json contenant les informations d’inscription se déclenche dans une fonction asynchrone dès que le bouton *Envoyer* est activé.

### Connexion

De la même manière que la page inscription, la page connexion est composée de la barre de navigation et d’un composant Login contenu dans le fichier *Login.js*. La fonction *render()* de Login est composée : d’un formulaire selon le même principe que le composant Connexion.

Une fois le json, contenant les informations de connexion, entré par un utilisateur, et la réponse du serveur reçue, deux évènements se déroulent :

* Dans le cas où la réponse du serveur rejette la demande de connexion, une pop up informe l’utilisateur du rejet, et laisse la possibilité à l’utilisateur de refaire une demande de connexion.
* Dans le cas où la réponse du serveur approuve la demande de connexion, une pop up informe l’utilisateur de l’approbation, et redirige l’utilisateur vers la page d’accueil dans le cas d’un utilisateur simple ou vers la page admin dans le cas d’un administrateur. Des informations telles que le token et le rôle de l’utilisateur, renvoyées par le serveur, ainsi que l’adresse mail et le nom de l’utilisateur, sont également stockées en local storage. Une variable associée à la bonne navigation de l’utilisateur dans l’application est également mise à jour.

### Sécurité

Lors de chaque appel au serveur, le token délivré par le serveur lors de la connexion est renvoyée avec la requête, sauf pour le chargement de la page Admin. Le token est valide tout le temps, jusqu’à ce que l’utilisateur active le bouton *Logout*. Lors d’un Logout, le local storage est entièrement vidé et l’utilisateur est redirigé vers la page de connexion.

Une vérification locale à lieu pour limiter l’accès à la page Admin à ceux dont le rôle le permet, ce qui expose l’accès à la page Admin par une modification manuelle du locage storage. Il faudrait mettre en place une requête qui se déclenche dès la construction du composant, s’assurant que le token qui envoie la requête est bien associé à un compte avec un rôle Admin.

### Profil

La page profil associé au composant du même nom affiche le profil de l’utilisateur dans des champs modifiables. Toute modification dans l’un de ces champs met à jour un json dans le state, qui sera renvoyé au serveur dès l’appui du bouton Modifier.

Un retour du serveur permet d’informer l’utilisateur de la modification ou non de son profil.

### MainPage

La page MainPage regroupe tous les groupes auxquelles l’utilisateur est rattaché. Deux champs pour créer et rejoindre un groupe sont également présent.

Une requête au serveur demandant les familles relatives à l’utilisateur identifié est envoyé avant que le composant soit monté. En fonction du résultat, un certain nombre de composant Famille sont créé. Le composant Famille est constitué d’un Bouton avec pour légende le nom d’un groupe et d’un texte représentant le code de ce même groupe.

Lorsque les boutons Créer et Rejoindre un groupe sont activés, une fonction asynchrone effectue un appel serveur correspondant. Une pop up avec le résultat de la requête informe l’utilisateur.

## Spécifications front-end mobile

## Spécifications back-end données échangées

## Spécifications back-end données utilisateurs