Analyse et Conception Objet de Logiciels

Projet Web

Aurélien VILMINOT
Damien CLAUZON
Hugo Elhaj-Lahsen
Nathan CERVEAU
Quentin BLASIAK

29 avril 2022

1 Analyse

1.1 Acteurs

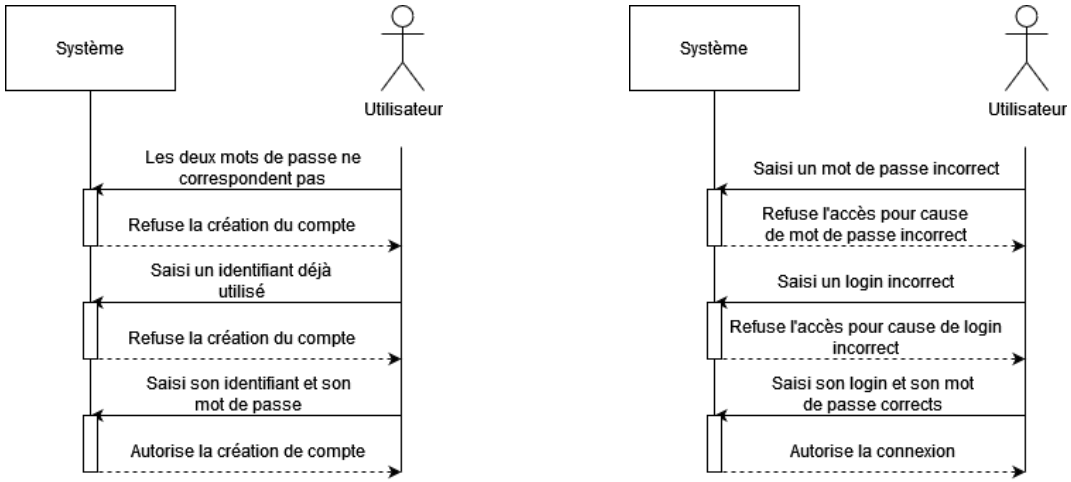
Dans le système réalisé, les acteurs présents sont :

- **Lecteur** : utilisateur de l'application sans authentification. Il peut lire des histoires et en avoir un historique local.
- **Utilisateur authentifié** : utilisateur possédant un compte. Il hérite du *Lecteur* et peut créer une histoire et sauvegarder son historique de façon permanente.
- **Collaborateur** : utilisateur étant affilié à une histoire. Il hérite d'*Utilisateur authentifié* et peut afficher les histoires qu'il peut modifier et créer un nouveau choix.
- **Auteur d'un paragraphe** : utilisateur étant affilié à la rédaction d'un paragraphe. Il hérite de *Collaborateur* et peut modifier le contenu, supprimer ou abandonner la rédaction d'un paragraphe.
- **Auteur d'une histoire** : utilisateur étant affilié à la rédaction d'une histoire. Il hérite de *Collaborateur* et peut définir le titre et le contenu du paragraphe initial. Concernant l'histoire, il peut en modifier sa visibilité, ajouter des collaborateurs et la rendre ouverte/fermée à la modification.

1.2 Diagramme de cas d'utilisation

Étant donné que le sujet du projet fourni des détails précis sur l'utilisation du système illustrés par des exemples, nous allons brièvement décrire les cas d'utilisation. Ces derniers sont accompagnés de diagrammes de séquence afin de visualiser les échanges avec le système.

1.2.1 Authentification



(a) Création d'un compte (b) Connexion

Figure 1 – Diagrammes de séquence : authentification

Pour accéder à l'application de manière authentifiée, l'utilisateur doit se connecter sur un compte existant ou en créer un le cas échéant. La création de compte impose une vérification du mot de passe ainsi qu'une unicité sur l' identifiant.

1.2.2 Lecture d'une histoire

Lors de la lecture d'une histoire, l'historique des choix par l'utilisateur est automatiquement enregistré localement. Il peut donc, à tout moment, revenir sur l'un de ces choix. Si le lecteur est authentifié, alors son historique peut être sauvegardé en ligne.

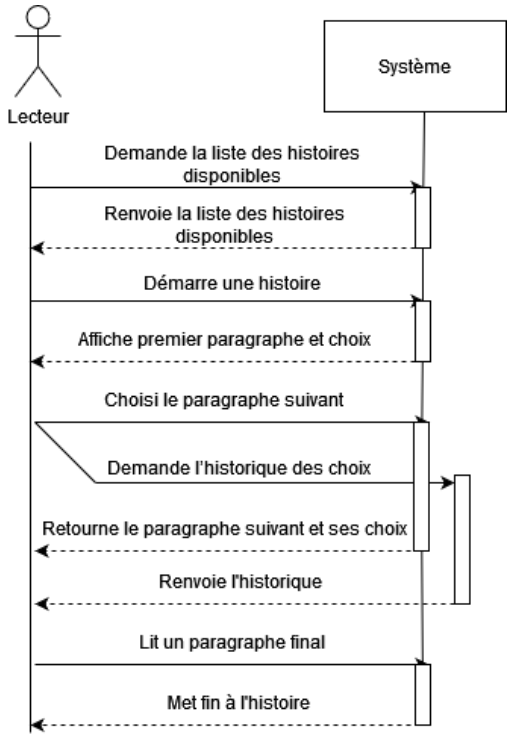


FIGURE 2 – Diagramme de séquence : lecture d'une histoire

1.2.3 Rédaction d'un paragraphe

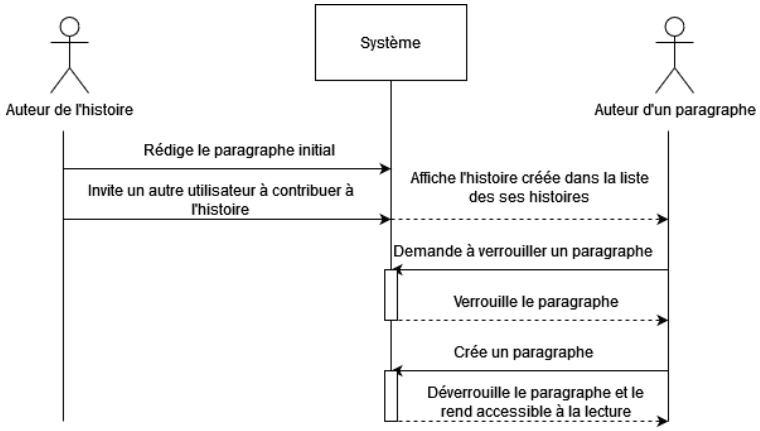


FIGURE 3 – Diagramme de séquence : rédiger un paragraphe

La création d'un paragraphe peut être réalisée par l'auteur de l'histoire ou l'un des collaborateurs qu'il a invité sur l'histoire. Lors de la création du paragraphe, son rédacteur doit spécifier son titre, son contenu, si c'est une conclusion ainsi que son paragraphe parent qui permet de l'atteindre. À noter que la démarche est similaire pour la création d'un nouveau choix entre deux paragraphes.

1.2.4 Modification d'un paragraphe

La modification d'un paragraphe ne peut se faire que via la modification de son contenu. Cette action est réalisable uniquement par son auteur.

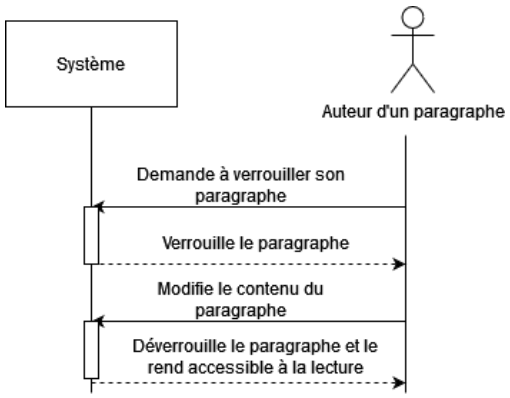


FIGURE 4 – Diagramme de séquence : modifier un paragraphe

1.2.5 Autres cas d'utilisation

Concernant les autres cas d'utilisation, un diagramme de séquence ne serait pas pertinent. En effet, par exemple, la création d'une histoire est possible pour n'importe quel utilisateur authentifié. Il n'y a donc pas d'intérêt à l'expliciter par un diagramme de séquence.

1.2.6 Diagramme de cas d'utilisation

Dans un soucis de lisibilité, le diagramme de cas d'utilisation est présent sur la page suivante.

Lecteur

Lire une histoire

"inclut"

Choisir un paragraphe

"étend"
Condition : des choix ont été faits

Consulter l'historique des choix

Se déconnecter

S'authentifier

"étend"
Condition : un compte a été créé

Se connecter

"étend"
Condition : ∅

Créer un compte

Utilisateur authentifié

Créer une histoire

Sauvegarder l'historique

Collaborateur

Afficher les histoires modifiables

Ecrire un nouveau choix

Créer un paragraphe

"inclut"

Donner un titre au nouveau choix

Créer un choix entre deux paragraphes

"inclut"

Choisir le paragraphe enfant

"inclut"

"inclut"

"inclut"

Ajouter une condition

Donner le statut final à un paragraphe

Choisir le paragraphe parent

{facultatif, et}

Auteur d'un paragraphe

Modifier contenu

Supprimer paragraphe

Abandonner un paragraphe

Auteur d'une histoire

Rendre publique/privée

Ajouter des collaborateurs

Rendre ouverte/fermée

Définir le titre

Rédiger paragraphe initial

1.3 Schéma relationnel

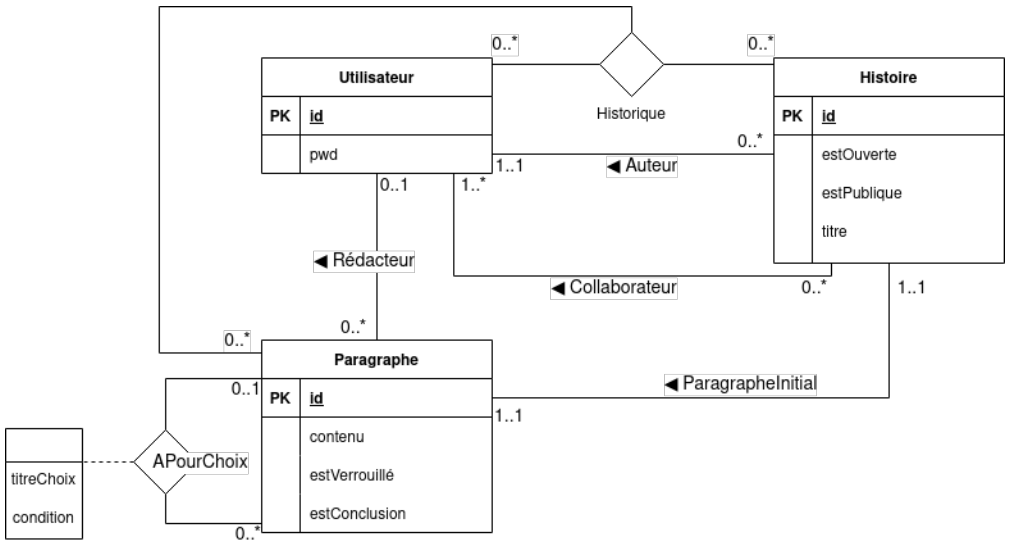


FIGURE 5 – Schéma Entité/Association

Le schéma utilisé pour la base de données n'est pas demandé mais nous avons jugé utile de l'intégrer dans ce document. En effet, il reflète toute l'analyse du sujet que nous avons faite et que vous avons transcrite pour faire fonctionner la base de données.

2 Conception

2.1 Architecture MVC

Nous avons pris le soin de nommer l'architecture des fichiers pour faire apparaître la structure MVC. La partie backend contient le Modèle et le Contrôleur, la partie frontend contient la Vue.

Les parties Modèle et Contrôleur communiquent localement dans le serveur, principalement en effectuant des requêtes ou des envoies de données du Contrôleur vers le Modèle.

La partie Vue communique avec la partie Contrôleur en effectuant directement des requêtes vers le serveur. Les requêtes sont envoyées par la partie services de la Vue, et sont gérées par la partie routes du Contrôleur. L'authentification est gérée par un système de middleware dans la classe router.

Le diagramme des classes est présent sur la page suivante.

Modèle

Tables

Histoire

+id: int
+titre: string
+estOuverte: boolean
+estPublique: boolean
+ParagrapheInitial: Paragraphe
+Auteur: Utilisateur

+addCollaborateur(Utilisateur): void
+getCollaborateur(): array_Utilisateur
+hasCollaborateur(Utilisateur): boolean
+setCollaborateur(array_Utilisateur): void
+getAuteur(): Utilisateur
+setAuteur(Utilisateur): void
+getParagrapheInitial(): Paragraphe
+setParagrapheInitial(Paragraphe): void

Paragraphe

+id: int
+contenu: string
+estVerrouille: boolean
+estConclusion: boolean
+Redacteur: Utilisateur

+getRedacteur(): Utilisateur
+setRedacteur(Utilisateur): void
+addChoix(Paragraphe): void
+getChoix(): array_Paragraphe
+countChoix(): int
+isRedacteur(Utilisateur): boolean
+hasPossibleChoix(): boolean
+leadToConclusion(): boolean
+updateState(): void

Utilisateur

+id: string
+pwd: string

+setHistorique(array_Historique): void
+getHistorique(): array_Historique
+clearHistorique(): void

Historique

+idUtilisateur: string
+idHistoire: int
+arrayParagraphe: array_JSON

ChoixTable

+ParagrapheId: int
+ChoixId: int
+titreChoix: string
+condition: int

Database

+database: Sequelize_object

+sync(options): void

Demande / Envoie données

Répond aux requêtes

Contrôleur

utilisateur

+registerUser(req, res): void
+loginUser(req, res): void
+whoami(req, res): void
+getUsers(req, res): void

paragraphe

+createParagraph(req, res): void
+getParagraph(req, res): void
+getPublicParagraph(req, res): void
+getChoiceList(req, res): void
+askToUpdateParagraph(req, res): void
+updateParagraph(req, res): void
+cancelModification(req, res): void
+deleteParagraph(req, res): void

histoire

+createStory(req, res): void
+updateStory(req, res): void
+getStory(req, res): void
+getPublicStory(req, res): void
+getUserStories(req, res): void
+getPublicStories(req, res): void
+getPublicAuthentifiedStories(req, res): void
+getParagraphList(req, res): void
+addCollaborator(req, res): void
+getCollaborators(req, res): void
+setCollaborators(req, res): void

historique

+saveHistory(req, res): void
+getHistory(req, res): void
+clearHistory(req, res): void

routes

router
histoire
historique
paragraphe
utilisateur
readOnly

Actions utilisateurs

Renvoie les données

Communication client-serveur

Vue

services

Story

+createStory(token: string, title: string, isOpened: boolean, isPublic: boolean): Histoire
+modifyStory(token: string, idStory: int, isOpened: boolean, isPublic: boolean): JSON
+getPublicStories(token: string): array_Histoire
+getPublicAuthentifiedStories(token: string): array_Histoire
+getUserStories(token: string): array_Histoire
+getParagraphList(token: string, idStory: int): array_Paragraphe
+getCollaborators(token: string, idStory: int): array_Utilisateur
+setCollaborators(token: string, idStory: int, collaborators: array_Utilisateur): JSON

Users

+register(username: string, password: string): JSON
+login(username: string, password: string): JSON
+whoami(token: string): JSON
+userList(token: string): JSON

Paragraph

+createParagraph(token: string, idStory: int, title: string, content: string, idParent: int, idChild: int, isConclusion: boolean, condition: int): Paragraphe
+getParagraph(token: string, idStory: int, idParagraph: int): Paragraphe
+getPublicParagraph(token: string, idStory: int, idParagraph: int): Paragraphe
+askToUpdateParagraph(token: string, idStory: int, idParagraph: int): Paragraphe
+updateParagraph(token: string, idStory: int, idParagraph: int, content: string): Paragraphe
+cancelModification(token: string, idStory: int, idParagraph: int): Paragraphe
+getChoiceList(token: string, idStory: int, idParagraph: int): array_Paragraphe

History

+saveHistory(token: string, idStory: int, arrayParagraph: array_Paragraphe): Historique
+getHistory(token: string, idStory: int): Historique
+clearHistory(token: string, idStory: int): Historique

views

Welcome
Login
Register
Home
Paragraph
UserStories
CreateStory
SetCollaborators
SetParagraphs
SetParagraph

App

components

multiSelect
popup
stories
paragraph

2.2 Conception détaillée

2.2.1 États d'un paragraphe

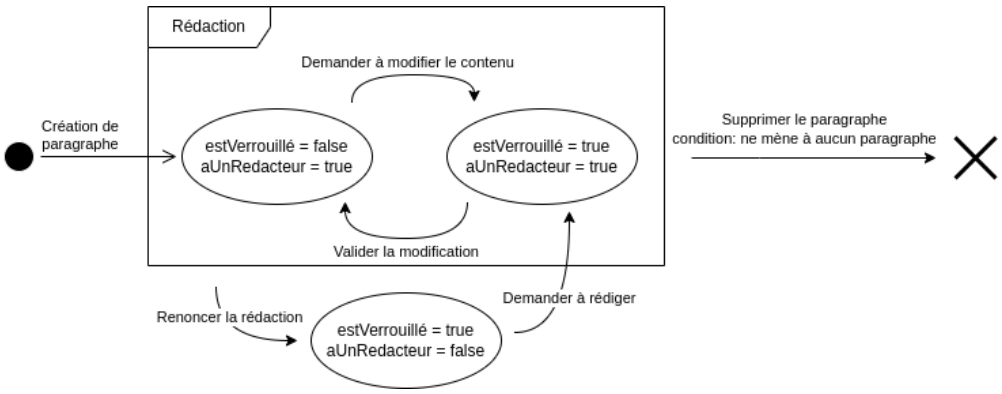


Figure 6 – Diagramme d'états-transition : états d'un paragraphe

La rédaction d'un paragraphe modifie certains états de ce dernier. Lorsqu'un utilisateur demande à modifier un paragraphe, ce dernier devient verrouillé si les conditions sont respectées. De plus, le paragraphe se voit associé un rédacteur. En revanche, si un rédacteur renonce à la rédaction d'un paragraphe, alors il devient verrouillé mais dépourvu d'auteur et peut donc être rédigé par un autre utilisateur.

3 Manuel Utilisateur

3.1 Installation de l'application

Afin d'installer l'application développée au cours de ce projet, deux possibilités s'offrent à l'utilisateur. La première se fait via une installation directe sur un smartphone Android à partir du fichier `.apk` fourni à la racine du répertoire GitLab (cf. dépôt). La deuxième possibilité d'installation peut se faire via l'émulation d'un terminal Android, disponible sur les ordinateurs de l'Ensimag.

3.1.1 Fichier `.apk`

Afin d'installer l'application sur un smartphone Android (version OS minimale : 11), ci-dessous la démarche à suivre :

1. Récupérer le fichier sur le dépôt GitLab
2. Autoriser l'installation de fichiers `.apk` sur le smartphone
3. Ouvrir le fichier `.apk` pour procéder à l'installation

3.1.2 Émulation sur ordinateur

Les instructions suivantes ne sont valides que sur les ordinateurs de l'Ensimag ou un ordinateur personnel possédant la configuration adéquate. La version de Node doit être la dernière en date. Ci-dessous, les instructions pour démarrer l'application sur un émulateur :

1. Démarrer le serveur local de l'application (commandes à exécuter en se plaçant à la racine du projet) :

```
> cd back/
> npm install
> npm run doc
> npm run create-db
> npm start
```

2. Dans un nouveau terminal, démarrer react native (commandes à exécuter en se plaçant à la racine du projet) :

```
> cd front/
> npm install
> npx react-native start
```

3. Dans un nouveau terminal, démarrer l'émulateur Android (commandes à exécuter en se plaçant à la racine du projet) :

```
> cd front/
> npx react-native run-android
```

3.1.3 Jeu de données

Il est possible d'insérer un jeu de données prédéfinies lors de l'émulation sur ordinateur. Pour cela, il faut exécuter les commandes suivantes en se plaçant à la racine du projet :

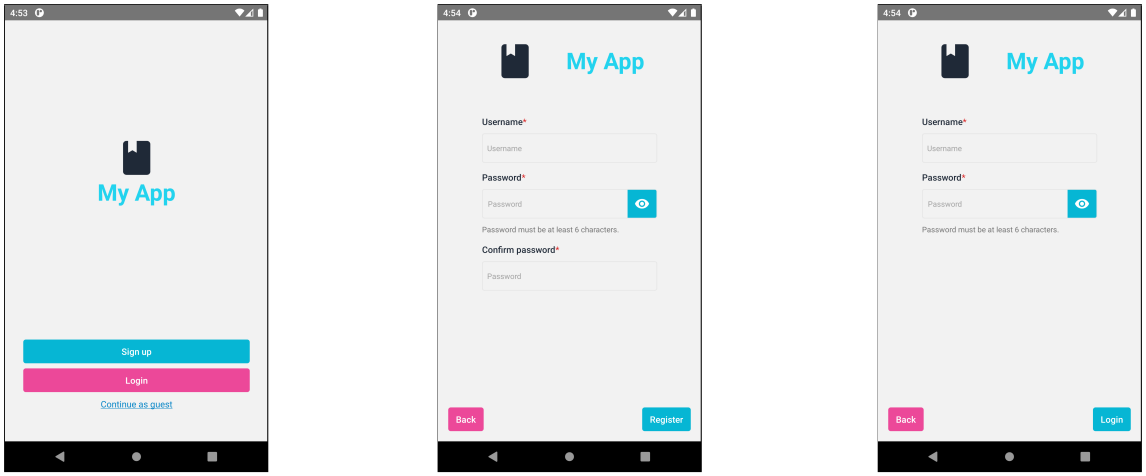
```
> cd back/
> npm run sample-db
```

Si l'application est directement installée sur un smartphone Android, elle est alors directement fonctionnelle mais vierge de toute donnée. C'est donc à l'utilisateur de créer un compte, de créer une histoire…

3.2 Utilisation de l'application

Une fois l'application installée, son utilisation est identique quelque soit le support d'exécution (terminal Android ou émulateur).

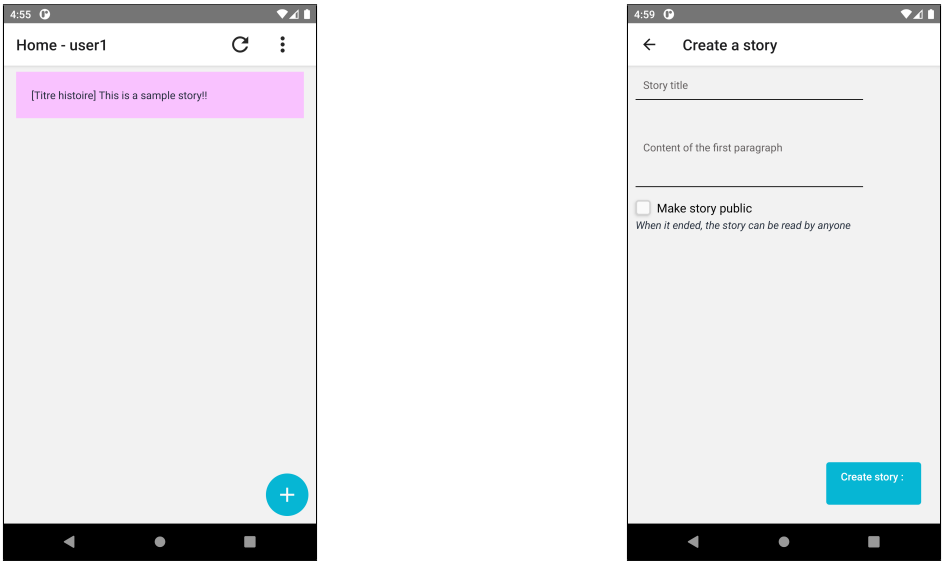
En premier lieu, l'utilisateur peut créer un compte, se connecter directement avec un compte existant (identifiant : *root* / Mot de passe : *azerty*) ou lire les histoires en mode invité.



(a) Accueil (b) Inscription (c) Connexion

FIGURE 7 – Pages d'accueil et de connexion/inscription

Une fois connecté, l'utilisateur peut créer une histoire en appuyant sur le bouton "+" en bas de l'écran d'accueil.



(a) Liste des histoires pouvant être lues (b) Création d'une histoire

FIGURE 8 – Liste des histoire et création d'une histoire

Une popup peut s'ouvrir en appuyant sur les trois point verticaux en haut à droite de la page d'accueil afin de consulter les histoires de l'utilisateur, de sa propre création ou celles dont il est collaborateur, ou se déconnecter.

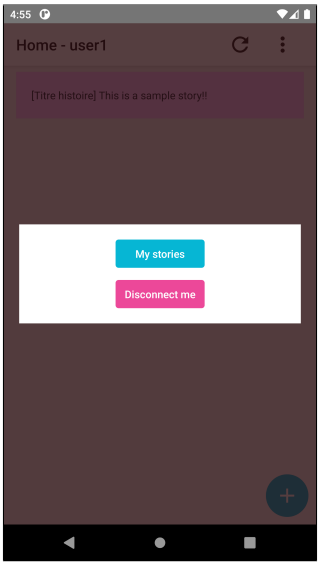


FIGURE 9 – Popup permettant de consulter les histoires de l'utilisateur ou de se déconnecter

Depuis la page des histoires de l'utilisateur, il est possible de modifier certains paramètres de l'histoire :
- Modifier la visibilité (publique/privée)
- Autoriser/Interdire la modification de l'histoire par les collaborateurs
- Modifier les collaborateurs
- Modifier les paragraphes

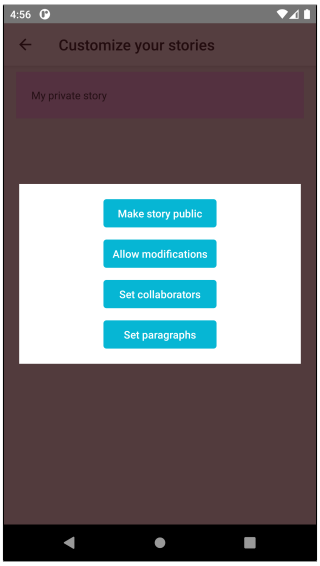
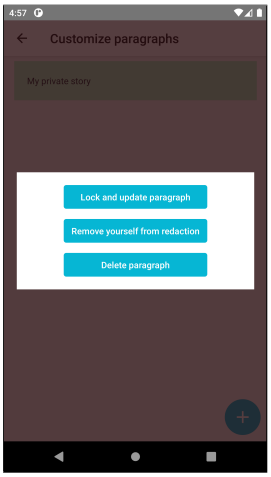


FIGURE 10 – Popup permettant de modifier une histoire spécifique

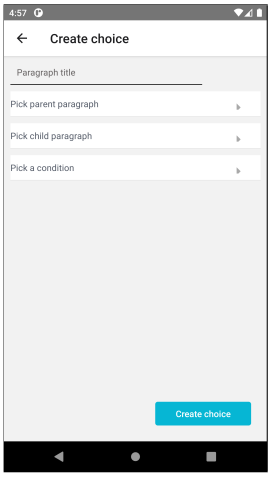
Lors de la modification des collaborateurs, il est possible de sélectionner/désélectionner tous les utilisateurs présents sur l'application. Il n'est pas possible pour un auteur de se retirer de cette liste.

FIGURE 11 – Spécification des collaborateurs d'une histoire

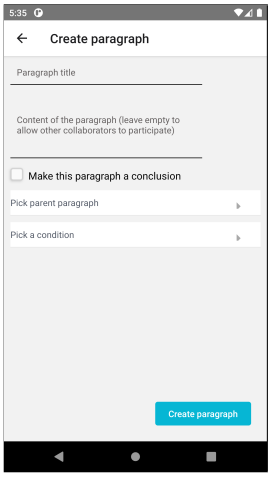
Enfin, il est possible de créer et/modifier des paragraphes d'une histoire. Lorsqu'un paragraphe est créé, seul son contenu peut être modifié. Il est également possible de se retirer de la rédaction d'un paragraphe ou de supprimer définitivement ce dernier si l'on en est le rédacteur. Depuis la page des paragraphes d'une histoire, l'utilisateur peut créer un nouveau paragraphe, en indiquant le paragraphe parent auquel il est lié, ou de créer un nouveau choix entre deux paragraphes existants. Ces deux dernières actions se font en remplissant le formulaire qui leur est associé.



(a) Popup permettant de modifier un paragraphe spécifique



(b) Création d'un choix



(c) Création d'un paragraphe

FIGURE 12 – Pages de création/modification de paragraphes et choix

4 Bilan

À l'issue de ce projet, nous pouvons donc tirer un bilan sur ce dernier. Tout d'abord, concernant les outils, nous avons principalement utilisé *Diagramme.net* (cf. lien) qui permet de réaliser tout type de diagrammes UML. De plus, l'outil est collaboratif car nous pouvions travailler à plusieurs en simultané sur un même diagramme. Dans un premier temps, nous avions utilisé le logiciel *StarUML*. Pour sauvegarder le fichiers créés, nous les mettions donc sur le dépôt GitLab. Cependant, cette solution est rapidement apparue comme peu pratique du fait de l'impossibilité de collaborer ensemble sur un même fichier. C'est pourquoi nous avons basculé sur l'outil *Diagramme.net*.

La réalisation de ce document nous a permis de mieux structurer notre projet. La partie de développement de ce dernier fut donc beaucoup fluide que sur des projets antérieurs. Que ce soit l'analyse ou la conception, ces phases ont permis de réduire la complexité du sujet tout en fournissant un fil conducteur pour la construction du système.