

Rapport de stage

Nathan GORON

Développement d'un module de questionnaire sur un système de gestion de contenu destiné aux acteurs culturels



Maître de stage

Vincent FRANCOISE



Responsable pédagogique

Florian BENAVENT

Soutenance :
20/06

Jury :
LECARPENTIER Jean-Marc
BENAVENT Florian

Licence 3 informatique

Année universitaire 2017-2018

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au bon déroulement de mon stage, tout particulièrement l'équipe de Twelve Solutions pour son accueil et sa bienveillance durant ses trois derniers mois.



Je remercie chaleureusement Mr.Francoise Vincent, développeur, co-gérant pour m'avoir accompagné en tant que tuteur et pour avoir su m'accorder une indépendance ainsi qu'une confiance mesurée tout en m'impliquant dans des projets concrets.

Je remercie également Mr.Perchey Yuri, Mr.Vincent Maxime et Mme.Cuisset Anne-flore pour les moments et les connaissances partagés durant ces trois mois.

Sommaire

1. Introduction.....	5
1.1 Cadre du stage.....	5
1.2 Environnement et outils.....	6
1.2.1 Concepts Vue.js nécessaires pour suite du rapport.....	7
2. Contexte et objectifs.....	8
2.1 Qu'est ce que le Wivi ?.....	8
2.2 Comment fonctionne le Wivi ?.....	8
2.3 Fonctionnalités.....	9
2.3.1 Application de gestion.....	9
2.3.2 Application de visite.....	10
3. Déroulement : missions et travail réalisé.....	11
3.1 Prise en main des outils.....	11
3.2 Application livre d'or.....	11
3.3 Mission de stage : module de quizz.....	12
3.3.1 Module de quizz dans le Wivi v1	12
3.3.2 Objectifs.....	13
3.3.3 Première version, identification des problématiques.....	14
3.3.4 Nouveaux types de questions.....	16
3.3.5 Nouvelles problématiques.....	17
3.3.6 Notion de déclenchement de la question.....	18
3.3.7 Découpage en composants.....	19
3.3.8 Liaison de médias, orientation WYSIWYG.....	19
3.3.9 Refonte graphique, stepper.....	20
3.3.10 Sauvegarde des questions depuis le CMS.....	21
3.3.11 Fin de stage, ajustements et utilisation externe des questions..	22
4. Conclusion	
4.1 Bilan.....	24
4.2 Glossaire.....	24
4.3 Annexes.....	24

1 Introduction

1.1 Cadre du stage :



Dans le cadre de la finalisation de ma licence informatique à l'université de Caen et pour une période de trois mois à compter du 27/03/2018, j'ai été accueilli au sein de Twelve Solutions, afin d'y accomplir un stage en développement web applicatif.

L'intégration d'une SSII, telle que Twelve Solutions, m'attirait tout particulièrement, de par son aspect très formateur sans nul doute du au besoin permanent de résultat : la pression générée dans le cadre d'une start-up à petit effectif numérique constituait selon moi un moteur performant (dans certaines limites) d'apprentissage de la programmation, de l'abstraction de problématique mais aussi de flexibilité dans les compétences.



Twelve Solutions est un bureau d'étude numérique travaillant principalement aux cotés de lieux culturels Français tels que les musées, au travers du développement d'application de gestion mais aussi via des projets numériques spécifiques à la demande.

L'entreprise compte 3 employés permanents dont deux profils techniques. Les applications proposées visent à l'amélioration et l'optimisation des visites de lieux culturels par le numérique : livre d'or interactif numérique, application de visite audioguidée, interface de gestion de contenu pour les administrateurs de musées ...

Bien que les ressources internes de la start-up la profile tout particulièrement pour le développement d'outils informatiques, Twelve Solutions propose également son propre produit de visite audioguidée des plages du débarquement de Normandie, la D-Day box.

L'équipe était donc constituée de :

Maxime Vincent : Développeur full-stack et **formateur l'IMIE**

Yuri Perchey : Co-gérant

Vincent Francoise : Co-gérant et développeur

Anne-Flore Cuisset : Stagiaire en développement web

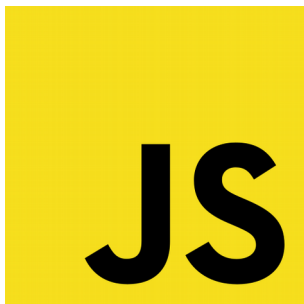
Lou Guilbert : Alternante chargée de communication

1.2 Environnement et outils

J'ai été autorisé à utiliser mon propre matériel durant ce stage et ai choisi d'utiliser Ubuntu afin de profiter d'un système rapide et efficace. Adeptes des IDE légers, j'ai opté pour l'utilisation de Visual studio code pour sa clarté et sa simplicité d'utilisation.

Les Bureaux étant séparés en deux pièces, il m'a été proposé de travailler seul dans le second bureau. J'ai cependant préféré privilégier l'immersion et suis resté dans les bureaux principaux afin d'échanger plus régulièrement sur mes avancées et sur les directions à prendre avec l'équipe.

Pour le développement de ses applications (et tout particulièrement pour la plate-forme sur laquelle j'ai eu l'occasion de travailler) les choix techniques arrêtés sont les suivants :



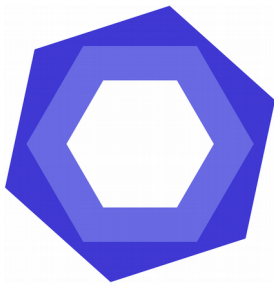
JavaScript : Langage de programmation et de scripting. Ici utilisé dans le cadre de Vue.js



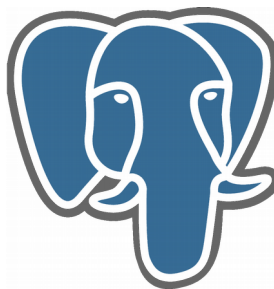
Vue.js : Vue.js est un Framework open-source permettant de construire des interfaces graphiques. Nous l'utiliserons dans ce projet comme moteur de création d'application web.



Vuetify : Vuetify est un framework pour Vue.js de construction d'interface web MaterialDesign (convention graphique établie par google)



Eslint : ESLint est un linter : il s'agit d'un outil de développement relevant les erreurs relatives au respect des conventions d'écritures de code javascript et vue.js



PostgreSQL : Base de données relationnelles

1.2.1 Concepts Vue.js nécessaires à la compréhension de la suite du rapport :

Composant : Un composant est un élément réutilisable possédant sa propre logique et son propre fonctionnement. Il permet de construire des applications sous forme de « blocs » et d'éviter les répétitions de code. Les composants sont construits en trois blocs : un bloc template composé de code HTML, un bloc script définissant la logique (fonctions) du composant et un bloc style pour agrémenter le template de css.

Store : Le store est un espace de stockage sous forme d'arbre définissant « l'état » d'une application. On y stocke dynamiquement les données nécessaires au rendu visuel de l'application en fonction de son état à un instant T.

2 Contexte et objectifs

Le but de mon stage était de réaliser un module de génération de question pour l'outil **Wivi** (plate-forme de gestion de contenu et guide de visite pour les musées). Une première version de ce module de quizz avait déjà été développée dans une version antérieure du Wivi, il s'agissait donc de reprendre les idées directrices du module, de repenser ses fonctionnalités et de les développer sur la nouvelle version du CMS¹.

2.1 Qu'est ce que le Wivi ?



Le Wivi est un outil visant à centraliser la gestion d'un établissement culturel, à catégoriser et valoriser son contenu via des outils variés afin d'apporter une plus-value durant l'expérience de visite.

2.2 Comment fonctionne le Wivi ?

Le système Wivi est composé de deux applications :



- Une application web de gestion de contenu destiné au gérant de musée, sur laquelle il peut référencer son contenu sous forme de points d'intérêts, le documenter et le valoriser via plusieurs fonctionnalités
- Une application web de visite destinée à accompagner les visiteurs pendant leur tour du musée sur leur smartphone. L'intéressé y retrouvera des informations supplémentaires sur le contenu du site ainsi que des activités variées améliorant son expérience de visite.

La première version de l'application (aujourd'hui archivée) était développée en Angular, la nouvelle version utilise le framework Vue.js précédemment présenté.

2.3 Fonctionnalités

2.3.1 Application de gestion

L'application de gestion est (au jour du début de stage) découpée en 8 modules interdépendants:

- le Dashboard : interface d'aperçu du musée, permettant de consulter des statistiques relatives aux comportements des visiteurs recueillis depuis l'application de visite, mais également d'établir les paramètres du CMS tels que les couleurs du thème ou l'activation/désactivation de certaines fonctionnalités.

- la section Utilisateur : Permet de gérer les droits d'accès à l'application








- le Gestionnaire de médias : interface de gestion des différents médias utilisés dans l'application de visite, elle permet de poster vidéos audios et images et est dotée d'une fonctionnalité de redimensionnement (cropper) afin de pouvoir utiliser une image dans différent contexte. Une fois envoyée sur l'application, les médias peuvent être liés à d'autres contenus tels que des points d'intérêts ou des quizz.

- Les collections : représente un ensemble de contenus. Une collection permet de regrouper des points d'intérêts afin de les catégoriser. On pourrait par exemple distinguer chaque pièce différents d'un musée dans des collections différentes

- Le gestionnaire de points d'intérêt : un point d'intérêt (POI) représente la description d'un contenu consultable. Elle peut être composée de texte, d'images, de vidéos, de pistes audios ou encore des liens menant vers d'autres POI à l'instar d'une page wikipedia.

- Les audioguides : pistes audios de description accompagnant les visiteurs

- Les personnalités : permet de référencer des personnalités importantes dans le contexte du lieu culturel et de les lier aux différents contenus. On liera par exemple un peintre à un ou plusieurs tableaux

	Dashboard
	Utilisateurs
	Gestionnaire de médias
	Collections
	Points d'interet
	Audioguide
	Personnalités

Menu de navigation du Wivi

2.3.2 Application de visite

Depuis l'application de visite, un visiteur peut accéder aux audioguides enregistrés précédemment sur le CMS, il peut s'enrichir d'informations supplémentaires sur une œuvre à mesure qu'il progresse dans sa visite en consultant la page de point d'intérêt correspondante.

3. Déroulement : missions et travail réalisé

Ma mission de stage consistait au développement d'un module sur le Wivi, on m'a néanmoins proposé de m'adapter aux nouveaux outils et de m'exercer via des tutoriels en ligne ainsi que sur des missions de plus petite envergure avant de foncer tête baissée dans un projet fruit de plusieurs centaines d'heures de travail à l'architecture complexe.

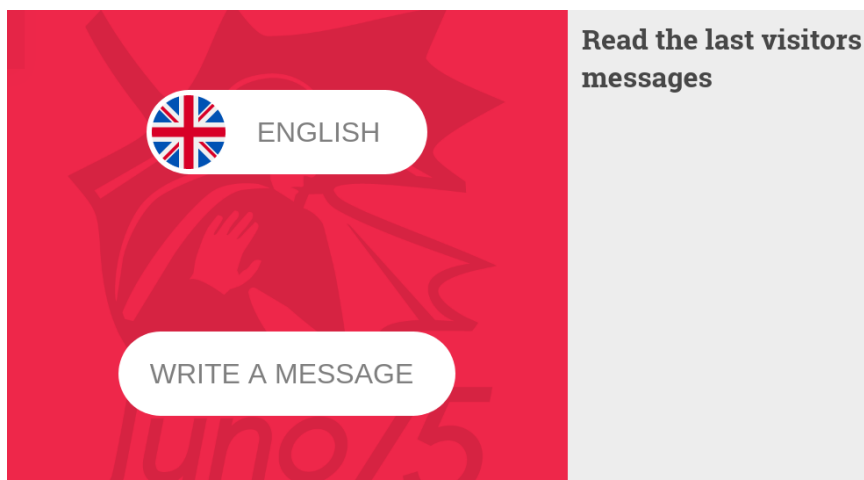
3.1 Prise en main des outils

A l'arrivée dans l'entreprise, le fonctionnement de Vue.js et Vuetify m'étant inconnu, on m'a proposé une série de tutoriels afin de pouvoir aborder au mieux mes missions une fois les outils bien en mains.

J'ai ainsi pu profiter de l'expérience de Vincent Francoise et de Maxime Vincent et ai pu m'imprégner de leur pratiques dans l'intention d'écrire mon code sans m'écarter de leurs habitudes de programmation et donc de faciliter la maintenance et le développement de mon travail.

3.2 Application livre d'or

A l'issue de mon introduction aux outils, l'équipe travaillait sur la finalisation d'une application de livre d'or numérique : Sur une borne composé d'un écran tactile et d'un NUC (petit ordinateur tout en un) un utilisateur devait pouvoir renseigner des informations tels que son age, sa nationalité ou bien une note sur 5 rendant compte de la qualité de sa visite le concernant puis laisser un message qui restera accessible à tous les autres utilisateurs sur la borne, à l'instar d'un véritable livre d'or



Visuel d'accueil de l'application

Cette application destinée aux acteurs culturels est composée d'une interface d'aperçu général à partir de laquelle le gérant peut monitorer les commentaires postés (les refuser ou les accepter selon leurs contenus afin d'éviter les abus), et de l'application de rédaction de commentaire présente sur la borne

Afin de finaliser l'application de gestion, il nous a été demandé, à moi et Anne-Flore, de réaliser une page de statistiques relatives aux informations laissées par les visiteurs sur la borne.

Nous avons pour ce faire utilisé les bases acquises en vue.js ainsi qu'une librairie de création de chartes statistiques : chart.js.

```
<v-card md3 width="30%">
  <p class="statistics">{{validatedComs}}</p>
  <v-card-title>
    <h5 class="title">Commentaires validés</h5>
  </v-card-title>
</v-card>

<v-card md3 width="30%">
  <p class="statistics">{{pendingComs}}</p>
  <v-card-title>
    <h5 class="title">Commentaires en attentes</h5>
  </v-card-title>
</v-card>

<v-card md3 width="30%">
  <p class="statistics">{{note}}<span class="statAverage">/5</span></p>
  <v-card-title>
    <h5 class="title">Note moyenne</h5>
  </v-card-title>
</v-card>
```

Extrait de code du livre d'or, exemple partiel de template en vue.js

3.3 Mission de stage : module de quizz

3.3.1 Module de quizz dans le Wivi V.1

Dans la première version du wivi, le module de quizz permet de créer une question lors de l'édition d'un POI. Une question est dès lors immuablement liée à un unique POI et ne concerne que le contenu de ce dernier.

Il n'existe dans cette version qu'un type de question: le QCM, ce type est caractérisé par la présence de plusieurs réponses différentes. L'utilisateur choisit tout simplement une ou plusieurs réponses écrites à une question écrite.

Les réponses aux question n'étaient pas sauvegardées. En effet, elles étaient stockées localement sur l'appareil du visiteur le temps de vérifier si ces dernières étaient correctes, puis elle étaient supprimée sans avoir été renvoyée au CMS pour être traitée.

Le module n'intégrait pas de notion de « déclenchement » des question (apparition des questions ou notion de « trigger ») : les questions était affichée explicitement dans un POI et ne dépendait d'aucun autre paramètre pour apparaître.

3.3.2 Objectifs

L'objectif de ce stage consistait donc à développer un nouveau module de quizz dans le CMS. Ce module devait permettre de générer des questions depuis le CMS et d'y répondre depuis l'application de visite en intégrant les notions suivantes :

- Notion de « triggers » (ou apparition des question) : les question doivent pouvoir apparaître en fonction de plusieurs types d'événement. Par exemple ou bout d'un certain temps passé sur l'application de visite ou bien après avoir consulté une vidéo dans un POI ...

- Intégrer plusieurs types de question

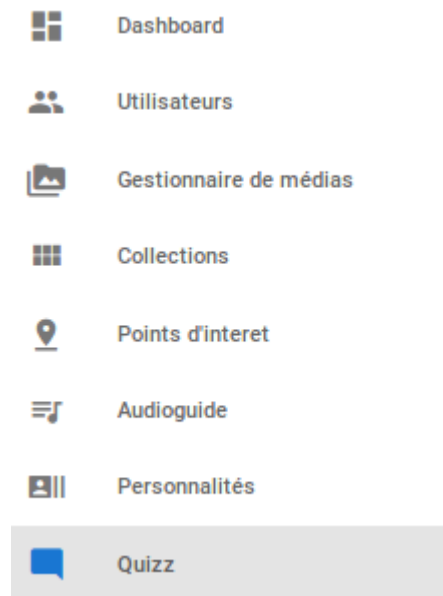
- Le musée doit pouvoir exploiter les réponses, cela implique de les récupérer lorsque l'utilisateur y répond.

- Supprimer la dépendance entre une question et un POI afin de pouvoir lier une question à un autre type de contenu ou bien lier une question à plusieurs POI différents

- Prévoir un type de question « ouverte » sans concept de bonne et mauvaise réponse afin de pouvoir recueillir des avis ou des commentaires de satisfaction de la part des visiteurs.

3.3.3 Première version, identification des problématiques

Une fois l'architecture du CMS comprise, j'ai pu créer le module et le référencer dans le menu avant d'étudier les problématiques relatives au développement



Il s'agit tout d'abord d'établir un objet de type « question ». On ne s'encombre pour l'instant pas des problématiques liées au stockage en base des questions ni du visuel de ces dernières, on se contentera simplement de stocker la question dans le store de l'application et de produire un éditeur de question fonctionnel. La majorité du travail à fournir se situe donc sur l'application de gestion de contenu.

On établit pour commencer un premier type de question : le type QCM. Ses caractéristiques sont simple :

- Une question sous forme texte
- Une ou plusieurs réponse possible jusqu'à 8 max afin d'éviter de surcharger le rendu visuel.

Pour pouvoir identifier une question unique, on attribue un id via la librairie **CUID**. Librairie permettant de générer des chaînes de caractères spécialement construites pour éviter au maximum les « collisions » (possibilité de générer deux id identiques extrêmement faible, même sur des solutions de stockage très importantes) tout en optimisant les performances de recherche.

On pense également à renseigner le type de la question afin de pouvoir l'identifier depuis l'application de visite et adapter le style de présentation en conséquences.

Pour une première ébauche, un exemple de l'objet « question » a donc pris la forme suivante :

```
question:
  id: "654qsdfs465f874s6q213f8qsd67f4qs5d32f1q",
  type: "QCM",
  question: "combien ai-je de doigts ?",
  answers:
    0:
      correct: true
      testData: "10"
    1:
      correct: false
      testData: "20"
```

Les champs « correct » définissent si les réponses 0 et 1 sont bonnes ou non.

Le pattern de l'objet étant établi, il s'agit maintenant de développer une première version visuelle du formulaire de création de question à l'aide de Vuetify :

question

combien ai-je de doigts

Ajouter une réponse

SUPPRIMER

10

☒ Bonne réponse

SUPPRIMER

20

☐ Bonne réponse

VALIDER

En l'état, l'outil permet donc de créer une question QCM composé d'un énoncé, d'un bouton permettant d'ajouter jusqu'à 8 réponses différentes, de blocs de réponses comprenant un bouton de suppression du bloc, d'un champs texte définissant la réponse et d'une tickbox définissant si la réponse est correcte.

3.3.4 Nouveaux types de questions

Maintenant que nous avons commencé à définir l'architecture d'une question et le visuel du formulaire de création nous avons pu nous pencher sur la création de nouveaux types de questions. Nous avons donc imaginé 3 nouveaux types de question :

-les questions vrai ou faux : Une question texte, et deux réponse prédéfinie non modifiable: Vrai ou Faux . Une question de ce genre aurait été facilement réalisable avec le type de question QCM, cependant l'ajout de nouveaux types de questions sert tant à faciliter la création des questions sur le CMS aux gérants de musées et à adapter le rendu visuel sur l'application de visite (les différents types de question profiteront chacune d'affichage différents)

-les questions numériques : Destinées a faire deviner une valeur numérique, elles sont définies par 4 champs : une borne minimale et maximale pour limiter les champs de recherche (que l'on ignorera si celles-ci sont laissée à 0 lors de la création d'une question), un pas définissant de combien en combien l'utilisateur fait progresser sa valeur et bien évidemment la bonne réponse.

Définissez les limites, le pas et la bonne réponse à la question

borne min 0	borne max 0	pas 1	Bonne réponse 0	
----------------	----------------	----------	--------------------	--

Premier aperçu des champs de création d'une question numérique

-les questions libres : Type de question particulière expressément demandée dans le cahier des charges, les questions libres n'ont pas de bonne réponse, elle sont simplement constituée d'une question et laisse le choix à l'utilisateur de rédiger ce qu'il veut. Ce type s'avérera utile pour des questions sur la satisfaction du client quant à sa visite par exemple (« qu'avez-vous pensé de votre visite ? »).

D'autres types de questions tels que des texte à trous ou encore des « chercher l'erreur » nous ont traversé l'esprit, mais nous nous sommes contentés de ces quatre premiers types afin de favoriser le « fonctionnel » au « complet », D'autant plus que ces types de question nécessiteraient beaucoup plus de travail que les 4 premiers déjà établis au niveau du rendu visuel pour que leur utilisation soit intuitive et ergonomique.

3.3.5 Nouvelles problématiques

En plus de nouveaux types de question il a été convenu que les questions devaient être argumentées de nouveaux contenus.

On introduit tout d'abord la notion de Commentaire : un commentaire sera un message d'explication relatif à la question qui s'affichera lorsque celui-ci y aura répondu afin de lui procurer encore plus d'informations utiles. On introduit donc dans l'objet de question un champs « comments » qui contiendra une liste de chaîne de caractère afin de pouvoir afficher plusieurs messages d'informations complémentaires .

On s'intéresse ensuite au cas où un utilisateur du CMS crée une grande quantité de question, le champs id étant une chaîne de caractère aléatoire permettait de distinguer une question d'une autre. Mais référencer une question en fonction d'un id originellement construit pour la comparaison par une machine la rend introuvable par un humain lorsque les questions commencent à s'accumuler. Il a donc fallu créer un champs supplémentaire « name » pour la question destiné à contenir une chaîne de caractère distinguable par un humain. En pratique dans le formulaire, cet information est un simple champs texte où l'on laisse libre choix à l'utilisateur de nommer sa question comme il veut du moment qu'il lui attribue bien un identifiant et que celui-ci n'est pas déjà attribué à une autre question :

Donnez un identifiant à votre question

Identifiant

Ma première question

Aperçu du champs d'identification sur le CMS

3.3.6 Notion de « déclenchement » de la question

Il a été établi dans le cahier des charges que les questions devraient parvenir aux visiteurs de plusieurs manières différentes sur l'application de visite: le type d'apparition le plus évident étant d'ouvrir un pop-up sur l'appareil de l'utilisateur et d'y dérouler la question, ce mode de fonctionnement bien qu'efficace s'avère également un peu intrusif : l'utilisateur n'a aucun contrôle sur l'apparition de la question et celle-ci pourrait le déranger voire l'agacer dans sa visite .

On définit dès lors un certain nombre de « trigger » c'est à dire de déclencheurs d'apparition de questions :

Le mode Pop-up : comme expliqué précédemment, la question s'affiche dans une sur-fenêtre lorsque l'utilisateur atteint un certain point dans une page.

Le mode Visible : la question est complètement déroulée dans le contenu d'un POI par exemple. Elle y est entièrement visible et attend simplement que l'utilisateur y réponde sans qu'il y soit pour autant obligé

Le mode Bouton : similaire au mode Pop-up, la question apparaîtrait sous forme de sur-fenêtre au dessus du contenu que l'utilisateur était en train de consulter. A l'exception près que la pop-up ne s'affiche que lorsque l'utilisateur a appuyé sur un bouton dédié.

Le mode Chronométré : La question s'affiche en Pop-up lorsque l'utilisateur a passé un certain temps à consulter un contenu.

Le mode Lié : l'apparition d'une question sera liée à certains événements spéciaux . Par exemple après avoir consulté 10 POI différents , ou bien après avoir regardé une vidéo ou encore suivi un audioguide ...

On ne s'intéresse pas encore à la façon dont les triggers seront gérés sur l'application de visite, on définit pour l'instant simplement un champ texte dans le formulaire précisant duquel de ces 5 types une question fait partie

3.3.7 Découpage en composants

Le module de quizz commence à prendre de l'ampleur, afin de respecter les conventions établies par vue.js et surtout pour éviter la répétition du code et favoriser la facilité d'écriture à l'avenir, il faut découper le module de quiz en plusieurs composants.

Maintenant que l'on a établi 4 types de questions différents et que chacun d'entre eux nécessite un style différent, on leur attribue à chacun un composant :

Standard.vue : composant de création des questions QCM

FreeAnswer.vue : composant de création des questions libres

TrueOrFalse.vue : composant de création des questions vrai ou faux

Numeric.vue : composant de création des questions numériques

Le principe de commentaires étant également implémenté dans 3 des 4 types de questions (les questions libres n'ont pas de réponses et ne nécessitent donc pas d'explications) On crée un composant Comments.vue afin de pouvoir facilement réutiliser ses fonctionnalités dans les trois composants de création de questions concernés sans avoir à écrire plusieurs fois le même code :

```
<Comments :question="question"></Comments>
```

code d'insertion du composant Comments dans un autre composant

On crée également un composant Statements.vue englobant la déclaration de l'intitulé de la question dans tous les différents types ainsi qu'un composant Answers.vue englobant la logique et le visuel (template et script) d'ajout de réponses dans les questions de type QCM.

3.3.8 Liaison de médias, orientation « WYSIWYG » du formulaire

Dans le module de quizz de l'ancienne version du wivi, il était possible de lier des médias (image, audio ou vidéo) aux intitulés de questions afin de les clarifier. Pour intégrer cette fonctionnalité, nous utilisons un composant déjà créé précédemment par Maxime Vincent destiné à un usage commun dans toute l'application : la Dropzone.

Le composant Dropzone permet d'afficher tous les médias de type images, audios ou vidéos et d'en sélectionner un en retournant son ID. Une fois l'id d'un média sauvegardé, on peut le retrouver facilement sur

l'application de visite). On utilise donc une Dropzone pour pouvoir attacher un ou plusieurs médias aux intitulés de questions, aux commentaires et aux réponses.

Ces trois champs prennent alors une orientation WYSIWIG, l'utilisateur pouvant ajouter à sa guise des champs texte ou des médias dans l'objet de question :

Ajoutez un commentaire qui s'affichera lorsque l'utilisateur aura répondu à la question

Un commentaire de type texte

DELETE

Ajouter un media

+TEXTE

+IMAGE

+AUDIO

+VIDEO

Formulaire d'ajout de commentaires avec la fonctionnalité de liaison de médias

L'objet question se complexifie. Les champs « Comments », Statements (= correspond à l'intitulé de la question) et Answers pouvant être agrémentés de médias, ils doivent être construits sous forme de liste afin de pouvoir contenir autant de champs que voulu :

```
comments:
  0:
    commentData: "blablabla un commentaire et une image"
    commentType: "text"
  1:
    commentData: "0368e64b342c9195278b8f790371288ae7593f95"
    commentType: "image"
```

Exemple d'un champ comments composé d'une question et d'une image

3.3.9 Refonte graphique, stepper

Le module de quizz commençant à s'alourdir, il devient difficile de distinguer les informations sur le formulaire (voir annexe II)

Pour faciliter la compréhension, on s'intéresse donc à l'utilisation d'un composant natif de Vuetify : le Stepper. Le stepper va permettre de décomposer en étapes la création de question et ainsi faciliter la compréhension (voir annexe III à VI)

3.3.10 Sauvegarde des questions depuis le CMS.

Jusqu'ici nous avons laissé de côté la sauvegarde des questions en base et nous étions contenté de les stocker dans le store de l'application. Maintenant que l'éditeur est fonctionnel, il s'agit maintenant d'envoyer les questions en bases de données afin qu'elle soit récupérable depuis l'application de visite. Pour cela, Maxime Vincent a configuré l'api et à créé 5 opérations de sauvegarde et de récupération vers l'API de développement :

- Une méthode GET pour récupérer la liste de toutes les questions en base
- Une méthode POST pour sauvegarder une question en base
- Une méthode GET pour récupérer une question en fonction d'un id
- Une méthode DELETE pour supprimer une question en fonction d'un id
- Une méthode PUT pour modifier une question en base

On utilise par la suite ces méthodes à la fois sur l'application de visite pour récupérer les questions, et sur l'application de gestion pour lister, supprimer, créer ou modifier les questions

L'envoi de question l'API étant régulée (la question doit respecter un schéma bien précis sans quoi la requête est rejetée), l'arborescence d'un objet question devait être le même quel que soit le type ou la configuration : toutes les questions suivent donc le schéma suivant (exemple ici dans le cas d'un QCM)

```
question:
  id: "654qsdfsq465f874s6q213f8qsd67f4qs5d32f1q"
  type: "QCM"
  identifier: "ma question QCM"
  statements: [{}]
```

```
  answers: [{}]
```

```
  comments: [{}]
```

Toutes les questions devront avoir un champs id de type string, un champs « type » de type string, un champs identifier de type string, et trois champs statements / answers et comments de type array.

3.3.11 Fin du stage, ajustements et utilisation externes des question

A l'approche de la fin du stage , il s'agit de finaliser les fonctionnalités actuelles de l'outil et préparer au mieux la base de code pour permettre son amélioration et son développement à l'avenir

On s'intéresse tout d'abord à l'internationalisation du contenu, le CMS étant consultable en Anglais comme en Français via un bouton situé en haut à droite de la page , le contenu doit être traduit automatiquement en fonction du choix de l'utilisateur. Pour se faire , on commence par déclarer tout les chaînes de caractères et contenu textuel dans un fichier spécifiquement dédié à la traduction :

```
export const messages = {
  en: {
    submit: 'Submit',
    cancel: 'Cancel',
    identifier: 'Identifier',
    question: 'Question',
    answers: 'Answers',
    clues: 'Clues',
    comments: 'Comments (facultative)'
  },
  fr: {
    submit: 'Valider',
    cancel: 'Annuler',
    identifier: 'Identifiant',
    question: 'Question',
    answers: 'Réponses',
    clues: 'Indices',
    comments: 'Commentaires (facultatif)'
  }
}
```

Objet « messages » contenant les traductions d'une page d'un formulaire de quiz

On importe ensuite un composant externe au module de quiz qui va permettre choisir la route à choisir dans l'objet messages en fonction des paramètres de langues choisis par l'utilisateur :

```
<v-stepper-step editable step="4">
  | {{t('comments')}}
</v-stepper-step>
```

utilisation du composant d'internationalisation

Dans un second temps, on intègre la notion d'indices aux formulaires de questions : il s'agira d'un texte agrémenté (ou non) de médias que l'utilisateur pourra choisir d'afficher ou non. En pratique, le code du composant d'indice ressemble beaucoup aux commentaires : un champ texte auquel on peut lier un ou plusieurs champs média.

Après réflexion sur les triggers, il a été établi que ces derniers devaient être séparés des questions et devenir une entité à part entière : en les décomposant, on laisse la possibilité de régir autres choses que des questions. On peut ainsi imaginer qu'un trigger puisse régir l'apparition d'un message plutôt que de le cantonner uniquement aux questions. Par exemple : lorsque l'horaire de fermeture du musée approche, un message automatique régité par un trigger vérifiant l'heure actuelle peut annoncer au visiteur qu'il doit se diriger vers la sortie.

On supprime par conséquent le stockage du trigger dans la question, un objet question devient donc complètement indépendant de tout autres type contenu. En somme, les questions ne sont donc utilisables pour l'instant que dans un POI en l'important directement comme un contenu au même titre qu'un texte ou qu'une image.

4. Conclusion

4.1 Bilan

Ce stage m'a permis d'approfondir mes compétences en Javascript, notamment en découvrant que le langage a bien plus à offrir que de la réactivité dans les pages web.

Même si ce stage ne s'inscrit pas tout particulièrement dans la suite de mon projet d'études, j'ai énormément appris autant d'un point de vue technique en développant mes compétences en programmation et en abstraction des problématiques, que d'un point de vue humain. Travailler dans une entreprise de petite envergure est selon moi une expérience unique, de par son aspect extrêmement formateur autant à la programmation qu'au sens des responsabilités et de l'autonomie.

4.2 Glossaire

CMS : Système de gestion de contenu. S'apparente dans ce rapport en au WIVI

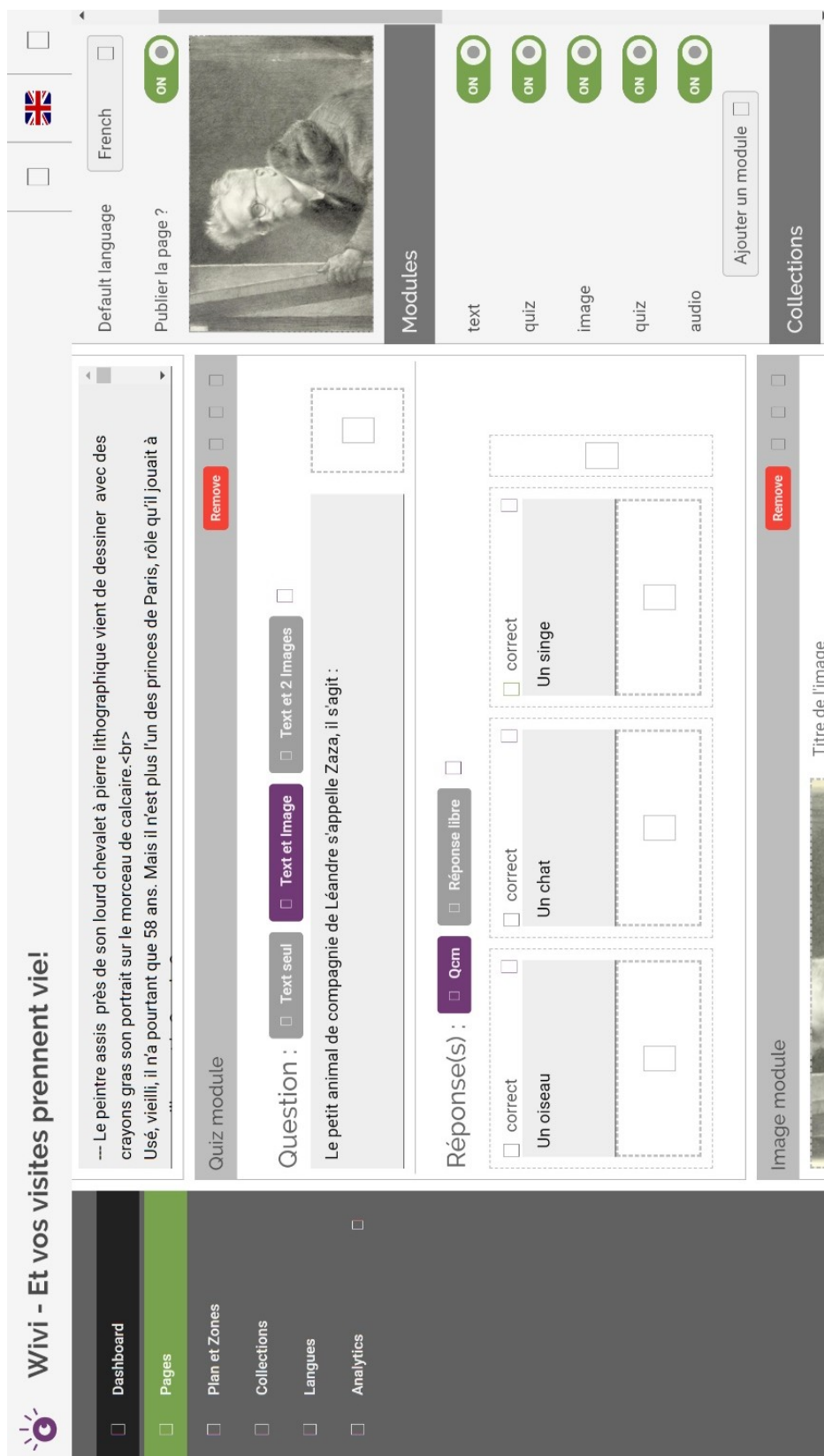
POI : Contraction de point d'intérêt, représente un contenu sur l'application de visite. Il s'agit en fait d'une page pouvant regrouper plusieurs type de contenus (du texte, des images, des vidéos, des audio, des liens vers d'autres POI et depuis le développement du module de quizz, des questions).

Triggers : Façon dont les question apparaissent sur l'application de visite

JS : Contraction de Javascript

4.3 Annexes

Annexe I : Visuel du module de quiz dans la V1 du WIVI en angular



Annexe II : Visuel général du formulaire de question avant refonte

Nouvelle question: MCQ

Donnez un identifiant à votre question

Identifiant

Rédigez la question

question

DELETE

Ajouter un media

+TEXTE

+IMAGE

+AUDIO

+VIDEO

Listes de toutes les réponses possible à la question



Ajouter une réponse

SUPPRIMER

SUPPRIMER

Lier un media

Lier un media

☐ Bonne réponse

☐ Bonne réponse

Ajoutez un commentaire qui s'affichera lorsque l'utilisateur aura répondu à la question

DELETE

Ajouter un media

+TEXTE

+IMAGE

+AUDIO

+VIDEO

VALIDER

ANNULER

Annexe III : Visuel du formulaire de QCM après refonte : identifiant

Nouvelle question: MCQ

1

Identifiant

Donnez un identifiant à votre question

Identifiant

2

Question

3

Réponses

4

Commentaires (facultatif)

VALIDER

ANNULER

Annexe IV: Visuel du formulaire de QCM après refonte : question

Nouvelle question: MCQ

1

Identifiant

2

Question

Rédigez la question

question

DELETE

Ajouter un media

+TEXTE

+IMAGE

+AUDIO

+VIDEO

3

Réponses

4

Commentaires (facultatif)

VALIDER

ANNULER

Annexe V : Visuel du formulaire de QCM après refonte : réponses

Nouvelle question: MCQ

1

Identifiant

2

Question

3

Réponses

4

Commentaires (facultatif)

Listes de toutes les réponses possible à la question

+

Ajouter une réponse

SUPPRIMER

Lier un media

☐ Bonne réponse

SUPPRIMER

Lier un media

☐ Bonne réponse

VALIDER

ANNULER

Annexe VI : Visuel du formulaire de QCM après refonte : commentaires

Nouvelle question: MCQ

1

Identifiant

2

Question

3

Réponses

4

Commentaires (facultatif)

Ajoutez un commentaire qui s'affichera lorsque l'utilisateur aura répondu à la question

DELETE

Ajouter un media

+TEXTE

+IMAGE

+AUDIO

+VIDEO

VALIDER

ANNULER

Annexe VII: Interface de gestion des questions sur le CMS :

