Université de Rouen

Rapport de projet de Langages Web 1

Quentin Brodier & Matthieu Coulon

Date : *23 Décembre 2015*

Table des matières

1 Description du projet

Pour ce projet de “Langages Web 1”, il nous a été demandé de développer une mini plate-forme web permettant de concevoir des quizs riches en ligne et d’y répondre et obtenir un score.

Les quizs contiennent :

- Un nom
- Une date de création
- Un type d’affichage pour le résultat final
- Une liste de questions (au moins une)
- Chaque question comprend entre 2 et 4 réponses, dont une seule est correcte.

Les questions et réponses peuvent contenir du texte non structuré (texte brut, texte en gras, texte en italique) ainsi que du code SVG et MathML.

Sur l’application, on distingue 3 rôles différents :

- **Invité :** l’invité est un utilisateur non enregistré qui accède à la plateforme. Cette personne peut consulter la liste de tous les quizs et y répondre afin d’obtenir un score. Il peut également récupérer l’URL d’un quiz, afin de le partager avec ses amis.

- **Utilisateur :** l’utilisateur est un membre qui s’est inscrit via l’accueil du site. A l’aide de ses identifiants, il peut créer ses propres quizs et les administrer. Il peut également choisir pour un quiz le format d’affichage final du résultat :
  - **1 :** Seul le score est affiché (nombre de réponses bonnes / nombre de questions)
  - **2 :** Le score est affiché ainsi que le résultat de la réponse à chaque question (correcte / incorrecte)
  - **3 :** Le score est affiché, le résultat de la réponse à chaque question (correct / incorrecte) ainsi que la bonne réponse à cette question.

- **Administrateur :** l’administrateur (unique) peut lister et supprimer les utilisateurs et les quizs.

Une démo de l’application est disponible en ligne à cette url : `http://178.62.73.248/`

Deux comptes sont créés (login / mot de passe) :

- Utilisateur simple : user / user
- Administrateur : admin / admin

2 Notice d'installation

Pour installer l'application, Il faut déposer le dossier Quizi à l'endroit où l'utilisateur souhaite que l'application soit placée, c'est-à-dire sur un serveur en ligne ou un serveur en local. L'application étant développé grâce à Symfony 2, elle utilise le module "Rewrite Module" d'Apache qu'il faut activer ainsi que les modules php suivants : "php_pdo_mysql", "php_intl", "php_xmlrpc", "php_mbstring".

2.1 Installation en local

Pour installer le projet en local, il faut configurer l'environnement d'accueil du projet, pour cela il faut créer un virtualhost.

2.1.1 Sous Windows

Pour cela, il faut éditer le fichier :
`C:/wamp/bin/apache/apache{version}/conf/extra/httpd-vhosts.conf`

Ajouter les lignes suivantes :

```
<VirtualHost *:80>
	ServerName localhost
	ServerAlias localhost
	DocumentRoot C:/wamp/www/
	<Directory "C:/wamp/www/">
		Options Indexes FollowSymLinks MultiViews
		AllowOverride all
		Require all granted
	</Directory>
</VirtualHost>

<VirtualHost *:80>
	ServerName quizi.dev
	ServerAlias www.quizi.dev
	DocumentRoot "C:/wamp/www/Quizi/web"
	<Directory "C:/wamp/www/Quizi/web">
		Options Indexes FollowSymLinks MultiViews
		AllowOverride all
		Require all granted
		Allow from all
	</Directory>
</VirtualHost>
```

A adapter selon votre situation :

```
80 : port du serveur apache
C:/wamp/www/ : adresse du dossier www de Wamp
quizi.dev : URL du projet
```

Il faut par la suite modifier (en tant qu'administrateur) le fichier host se trouvant :
`C:/Windows/drivers/etc/hosts`

Ajouter les lignes suivantes :

```
127.0.0.1    quizi.dev
::1          quizi.dev
```

Pour finir, il faut relancer le serveur apache et se rendre à la dernière étape d'installation (2.1.3).

2.1.2 Sous Mac / Linux

Les opérations sous Linux sont similaires à celles sous Mac, seul les chemins d'accès des fichiers sont différents.

Pour cela, il faut éditer le fichier :
`Macintosh/Applications/MAMP/conf/apache/httpd.conf`

Enlever le # (pour dé-commenter) à la ligne se situant sous # Virtual Host :
`# Include /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf`

Enregistrer, puis aller éditer le fichier :
`Macintosh/Applications/MAMP/conf/apache/extra/httpd-vhosts.conf :`

Ajouter à ce fichier les lignes suivantes :

```
<VirtualHost *:80>
      DocumentRoot /Applications/MAMP/htdocs/
      ServerName localhost
</VirtualHost>
<VirtualHost *:80>
      DocumentRoot  /Applications/MAMP/htdocs/Quizi/web
      ServerName quizi.dev
      ServerAlias *.quizi.dev
      <Directory "/Applications/MAMP/htdocs/Quizi/web">
            Options Indexes FollowSymLinks MultiViews
            AllowOverride All
            Order allow,deny
            Allow from all
      </Directory>
</VirtualHost>
```

A adapter selon votre situation :

```
80 : port du serveur apache
/Applications/MAMP/htdocs/ : adresse du dossier htdocs de Mamp
quizi.dev : URL du projet
```

Il faut par la suite modifier (en tant qu'administrateur) le fichier host se trouvant : `Macintosh/etc/hosts`

Ajouter les lignes suivantes :

```
127.0.0.1    quizi.dev
::1          quizi.dev
```

Pour finir, il faut relancer le serveur apache et se rendre à la dernière étape d'installation (2.1.3).

2.1.3 Dernière étape d'installation

La dernière étape consiste à installer Composer pour mettre à jour les dépendances de l'application :

Ouvrez un bash / terminal à la racine du projet Quizi/ puis exécuter la suite de commandes :

```
curl -sS https://getcomposer.org/installer | php
php composer.phar update -o
```

Il faut ensuite exécuter le fichier demarrer.php afin de configurer la base de données. Il faudra saisir quelques informations : hôte, port, nom de la base de données, nom de l'utilisateur et mot de passe de l'utilisateur. Des données sont proposées par défaut (entre crochet), il suffit juste d'appuyer sur Entrée si vous souhaitez garder ces données.

```
php demarrer.php
```

Pour finir, il faut vider le cache afin de rendre l'application disponible :

```
php app/console cache:clear --env=prod --no-debug
```

L'application est maintenant installée. Utiliser l'URL spécifiée dans le virtual host pour y accéder. Deux comptes sont créés de base :

Compte user après inscription :

Login : user

Password : user

Compte administrateur :

Login : admin

Password : admin

2.2 Installation sur un serveur distant

2.2.1 La connexion en SSH sur le serveur est possible

Il faut se connecter en ssh via un terminal sur son serveur distant (ci celui-ci le permet, souvent le cas des VPS). On transfère avant tout l'archive du projet, puis on exécute la même procédure que l'installation 2.1.3. Il faut cependant ajouter un .htaccess permettant une bonne gestion des URL (éviter le /web sur le projet Symfony). Il sera aussi possible de créer des VirtualHost comme pour l'installation locale.

2.2.2 La connexion en SSH sur le serveur n'est pas possible

Certains serveur ne permettent pas un accès direct, un dépôt de fichier est juste disponible en FTP. C'est pour cela qu'un Bundle a été créer, ConsoleBundle. Il permet de générer une console en navigateur. Celle ci permet par la suite d'effectuer les différentes tâches de configuration et d'installation, comme ci vous étiez sur un terminal en console.

Pour plus de détail voir :

https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/utiliser-la-console-directement-depuis-le-navigateur

3 Technologies utilisées

Nous avons décidé d'utiliser les langages vus durant le cours de "Langages Web 1", c'est à dire : HTML, CSS (Sass), PHP, Javascript.

Pour cela, nous utilisons le framework **Symfony** dans sa version 2.7.6 de part son architecture MVC (Model View Controller) et sa popularité en France.

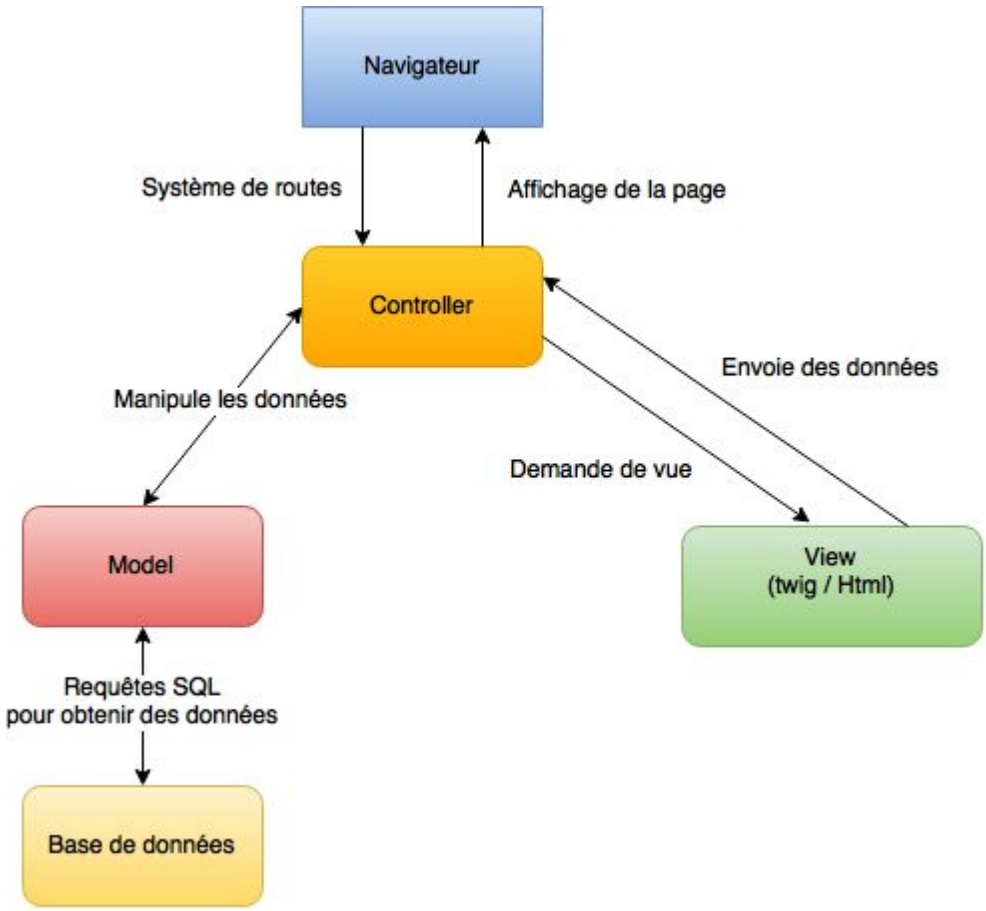

Figure 1 : Modélisation de l'architecture MVC avec Symfony

Symfony utilise :

- Pour les contrôleurs et les modèles : PHP Objet
- Pour les vues : le moteur de template Twig
- Pour la base de données : Doctrine

Pour personnaliser les vues avec le CSS et SASS, nous utilisons **Materialize**, un framework front-end moderne basé sur le Material Design de Google+.

4 Architecture Logicielle

4.1 Architecture du projet Symfony

Le framework Symfony dispose d'une architecture assez précise mais bien organisée afin de respecter le modèle MVC et d'intégrer d'autres principes. Il est composé de différents fichiers de paramètres et de dossiers, dont voici les principaux :

- **app**
  - **appKernel.php** sert à inclure les bundles dans le projet.
  - **config**
    - **parameters.yml** : configuration de la base de données
    - **routing.yml** : fichier permettant de définir les principales routes du système (un bundle détient son propre fichier de routing)
    - **security.yml** : permet de définir les paramètres de sécurité, notamment pour les utilisateurs et les restrictions sur le site (contrôle d'accès).
  - **resource :**
    - **view :**
      - **base.html.twig** : fichier html/twig de base du site, c'est celui qui contiendra les autres affichages du site.
- **src** : contient les différents Bundles du projet
  - **AppBundle :** bundle de base du site
  - **Bundle 2 :**
    - **controller :** contient les fichiers controllers du bundle
    - **entity :** contient les fichiers models du bundle
    - **resources**
      - **config :** (equivalent à ceux du dossier principal App)
        - **routing.yml**
        - **services.yml**
      - **views :** contient les fichiers d'affichages html.twig du bundle
- **vendors :** contient les bundles extérieurs pouvant être important (ex : twig, doctrine etc…)
- **web :** est le dossier "public" du site
  - **app.php** : fichier "index.php" du site
  - **app_dev.php :** identique à app.php sauf qu'il est utilisé pour le développement.
  - **js** : contient les fichiers javascript
  - **css** : contient les fichiers CSS
  - **images** : contient les images du site

Code couleurs : ***dossier*** - ***fichiers***

Le dossier **src** contient ce qu'on appelle des bundles. Les bundles sont tout simplement des sortes de "briques" d'un projet. Symfony2 utilise ce concept novateur qui consiste à regrouper dans un même endroit, le bundle, tout ce qui concerne une même fonctionnalité. Dans notre application, nous avons deux bundles : QuiziBundle (gestion des quizs) et UserBundle (gestion des users).

4.2 Architecture de la base de données avec Doctrine

Symfony intègre le module ORM Doctrine pour gérer le mapping entre la base de données et les modèles. Lorsque l'on créer une entité avec Doctrine (à l'aide de la console), Symfony créer automatiquement la base de données. Dans notre projet, nous n'utiliserons jamais de SQL et aucun SGBD comme PHPMyAdmin par exemple. Toute opération s'effectue en PHP objet avec Doctrine.

Nos modèles sont donc liés à la base de données. Un modèle Quiz par exemple aura dans la BDD une table Quiz, avec comme champs les attributs du modèle.

Voici donc le diagramme de classe associé aux modèles :

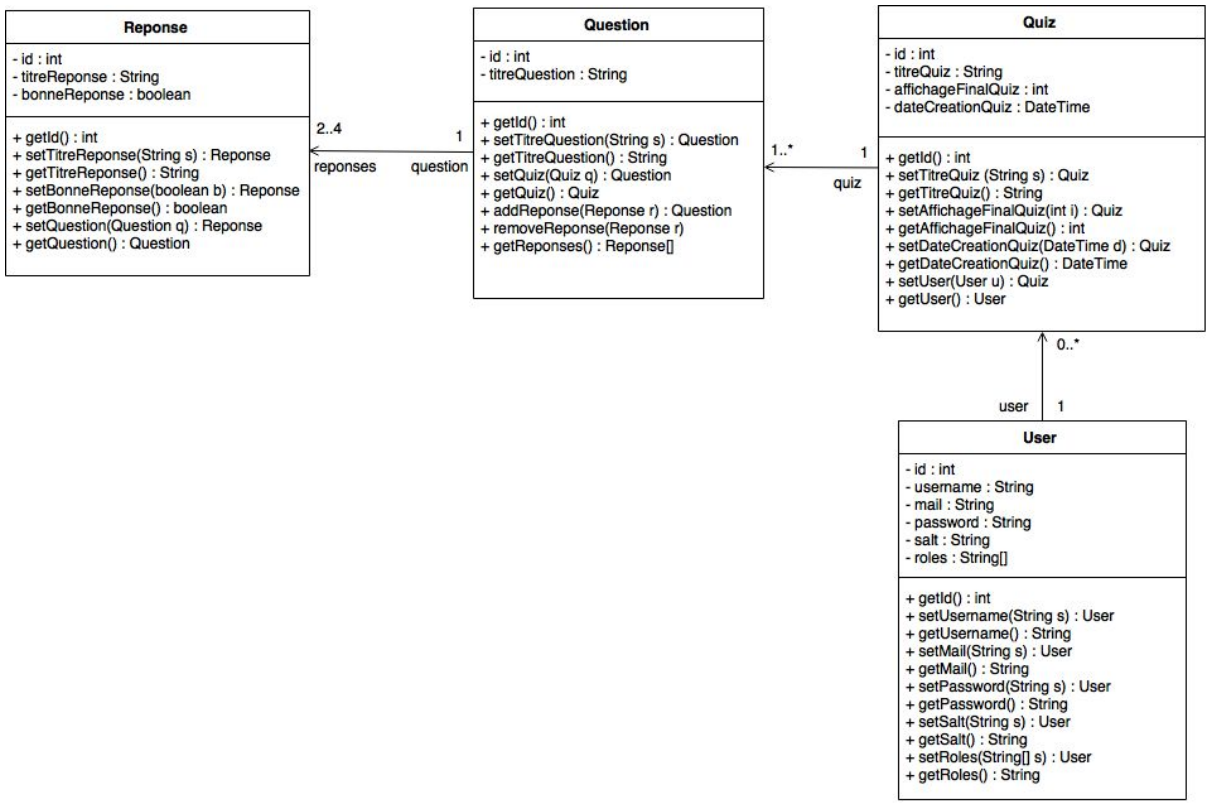

Figure 2 : Diagramme de classe des modèles

5 Documentation des classes et fonctions

5.1 Les modèles

5.1.1 User :

Attributs :

- **id (integer)** : clé primaire unique pour chaque objet
- **username (string)** : pseudo de l'utilisateur
- **mail (string)** : email de l'utilisateur
- **password (string)** : mot de passe de l'utilisateur (crypté)
- **salt (string)** : sel associé au mot de passe
- **roles (array)** : type de role de l'utilisateur (ROLE_USER ou ROLE_ADMIN)

Méthodes :

- **getId() : integer** : retourne l'id du user
- **setUsername($username) : User** : modifie le nom du user
- **getUsername() : String** : retourne le pseudo du user
- **setMail($mail) : User** : modifie l'email du user
- **getMail() : String** : retourne l'email du user
- **setPassword($password) : User** : modifie le mot de passe du user
- **getPassword() : String** : retourne le mot de passe du user
- **setSalt($salt) : User** : modifie le sel du user
- **getSalt() : String** : retourne le sel du user
- **setRoles($roles) : User** : modifie le rôle du user
- **getRoles() : array** : retourne le rôle du user

5.1.2 Quiz :

Attributs :

- **id (integer)** : clé primaire unique pour chaque objet
- **titreQuiz (string)** : nom donné au quiz
- **affichageFinalQuiz (integer)** : type d'affichage de résultat final du quiz
- **dateCreationQuiz (DateTime)** : date de création du quiz
- **user (User)** : utilisateur ayant créé le quiz

Méthodes :

- **getId() : integer** : retourne l'id du quiz
- **setTitreQuiz($titreQuiz) : Quiz** : modifie le titre du quiz
- **getTitreQuiz() : String** : retourne le titre du quiz
- **setAffichageFinalQuiz($affichageFinalQuiz) : Quiz** : modifie l'affichage final du quiz
- **getAffichageFinalQuiz() : integer** : retourne le numéro de l'affichage final du quiz
- **setDateCreationQuiz($dateCreationQuiz) : Quiz** : modifie la date de création du quiz
- **getDateCreationQuiz() : DateTime** : retourne la date de création du quiz
- **setUser($user) : Quiz** : modifie le user ayant créé le quiz
- **getUser() : User** : retourne le user ayant créé le quiz

5.1.3 Question :

Attributs :

- **id (integer)** : clé primaire unique pour chaque objet
- **titreQuestion (text)** : nom de la question
- **quiz (Quiz)** : quiz auquel appartient la question
- **reponses (Array<Reponse>)** : liste des réponses de la question

Méthodes :

- **__construct()** : constructeur qui initialise le tableau de réponses à la question
- **getId() : integer** : retourne l'id du quiz
- **setTitreQuestion($titreQuestion) : Question** : modifie le titre de la question
- **getTitreQuestion() : String** : retourne le titre de la question
- **setQuiz($quiz) : Question** : modifie le quiz auquel appartient la question
- **getQuiz() : Quiz** : retourne le quiz auquel appartient la question
- **addReponse($reponse) : Question** : ajoute la réponse à la question
- **removeReponse($reponse) : Question** : supprime la réponse à la question
- **getReponses() : Array<Reponse>** : retourne la liste des réponses de la question

5.1.4 Reponse :

Attributs :

- **id (integer)** : clé primaire unique pour chaque objet
- **titreReponse (text)** : contenu de la réponse
- **bonneReponse (boolean)** : booléen pour savoir si la réponse est correcte ou non
- **question (Question)** : question auquel appartient la réponse

Méthodes :

- **getId() : integer** : retourne l'id du quiz
- **setTitreReponse($titreReponse) : Reponse** : modifie le contenu de la réponse
- **getTitreReponse() : String** : retourne le contenu de la réponse
- **setBonneReponse($bonneReponse) : Reponse** : modifie le booléen de la réponse correcte ou non
- **getBonneReponse() : boolean** : retourne le booléen pour savoir si la réponse est correcte ou non
- **setQuestion($question) : Reponse** : modifie la question auquel appartient la réponse
- **getQuestion() : Question** : retourne la question auquel la réponse appartient

5.2 Les contrôleurs

5.2.1 DefaultController :

Méthodes :

- **indexAction()** :

Cette méthode est appelée par la route " `/` " et permet d'accéder à la page d'accueil si l'utilisateur n'est pas connecté ou la page des quizs si l'utilisateur est connecté

5.2.2 QuizsController :

Méthodes :

- **indexQuizAction()** :

Cette méthode est appelée par la route " `/quizs` " et affiche la liste de tous les quizs disponible auquel on peut participer en cliquant sur l'un d'eux en particulier.

- **viewQuizAction($idQuiz) :**

Cette méthode est appelée par la route " `/quizs/{idQuiz}` " et permet d'afficher une page pour le quiz caractérisé par l'id $idQuiz et un bouton pour démarrer la réponse au quiz. L'url peut être copiée afin de partager le quiz avec quelqu'un d'autre.

5.2.3 AdminQuizsController :

Pour toutes les routes suivantes, si un utilisateur non connecté tente d'accéder à une route non autorisée, il sera automatiquement redirigé sur l'espace de connexion, où il devra se connecter avant d'être redirigé sur la route voulue.

Méthodes :

- **indexAction()** :

Cette méthode est appelée par la route “ `/adminQuiz` ” et permet de lister les quizs afin de pouvoir les modifier. Si la personne connectée est un utilisateur basique, il récupère la liste de ses quizs qu'il peut modifier ou supprimer. Si la personne connectée est l'administrateur, il récupère la liste de tous les quizs qu'il peut supprimer.

- **ajoutAction(Request $request) :**

Cette méthode est appelée par la route “ `/adminQuiz/ajout` ” et permet d'ajouter un quiz. Une première partie de la méthode consiste à créer le formulaire d'ajout pour ensuite l'afficher dans le navigateur à l'aide de la vue. La seconde partie permet de récupérer les données une fois le formulaire saisi (à l'aide du paramètre $request) et de faire quelques vérifications avant d'ajouter le quiz dans la base de données.

- **supprimerQuizAction($idQuiz) :**

Cette méthode est appelée par la route “ `/adminQuiz/suppr/{idQuiz}` ” et permet de supprimer un quiz caractérisé par l'id $idQuiz dans la base de données. La méthode vérifie que le quiz en paramètre existe bien et qu'il appartient bien à l'utilisateur connecté (sauf si c'est l'administrateur).

- **modifQuizAction($idQuiz, Request $request) :**

Cette méthode est appelée par la route “ `/adminQuiz/modif/{idQuiz}` ” et permet de modifier un quiz caractérisé par l'id $idQuiz. La méthode vérifie que le quiz en paramètre existe bien et qu'il appartient bien à l'utilisateur connecté. Si l'administrateur est connecté et qu'il entre à la main l'URL d'une modification de quiz, ce dernier est automatiquement redirigé sur l'espace administration de quizs. Dans cette méthode, nous créons un formulaire de modification qui sera envoyée dans la vue. Une deuxième partie de la méthode consiste à récupérer les données une fois le formulaire saisi (à l'aide du paramètre $request). Si toutes les données sont valides, on met à jour le quiz. L'utilisateur peut modifier le quiz en modifiant ses informations, en ajoutant une question, en mettant à jour une question avec ses réponses, ou en supprimant une question.

- **addOrUpdateQuestion($quiz, $data) :**

Cette méthode permet d'ajouter ou de mettre à jour une question d'un certain quiz dans la base de données.

- **modifierQuestionsAction($idQuiz, $idQuestion, Request $request) :**

Cette méthode est appelée par la route “ `/adminQuiz/modif/{idQuiz}/que/{idQuestion}` ” et permet de mettre à jour les données d'une question (son titre et ses réponses) d'un quiz. Cette méthode vérifie que la question existe bien, qu'elle appartient au bon quiz qui appartient au bon utilisateur. Si l'administrateur est connecté et qu'il entre à la main l'URL d'une modification de question, ce dernier est automatiquement redirigé sur l'espace administration de quizs. Cette méthode créée un formulaire de modification d'une question et le renvoi à la vue correspondante. De

plus, elle récupère les données grâce à l'objet $request lorsqu'un utilisateur modifie une question.

- **supprimerQuestionsAction($idQuiz, $idQuestion, Request $request) :**

Cette méthode est appelée par la route " `/adminQuiz/modif/{idQuiz}/supr/{idQuestion}` " et permet de supprimer une question dans un quiz. Cette méthode vérifie que la question existe bien, qu'elle appartient au bon quiz qui appartient au bon utilisateur. Si l'administrateur est connecté et qu'il entre à la main l'URL d'une suppression de question, ce dernier est automatiquement redirigé sur l'espace administration de quizs.

- **regexScript($text) :**

Cette méthode permet de vérifier qu'une saisie ne comprend pas de balise <script>

5.2.4 ReponseController :

Méthodes :

- **repQuizAction($idQuiz) :**

Cette méthode est appelée par la route " `/quizs/rep/{idQuiz}` " et permet de participer à un quiz, en répondant aux différentes questions et d'obtenir un résultat dans la vue associée. Cette méthode vérifie avant tout que l'id $idQuiz est bien existant, et créé ensuite un formulaire de saisie des réponses. Dans la deuxième partie, cette méthode effectue le résultat en fonction des saisies de l'utilisateur.

5.2.5 SecurityController :

Méthodes :

- **loginAction(Request $request) :**

Cette méthode est appelée par la route " `/login` " et permet de connecter un utilisateur à la plateforme où de l'informer des mauvais identifiants saisis suite à la requète $request.

- **inscriptionAction(Request $request) :**

Cette méthode est appelée par la route " `/inscription` " et permet d'inscrire un utilisateur à la plateforme, après avoir vérifié que toutes les données saisies sont correctes et que le login et le mail n'existent pas déjà dans la base de données.

- **gestionUserAction() :**

Cette méthode est appelée par la route " `/gestion_utilisateur` " et permet de lister les utilisateurs afin de pouvoir les supprimer. Cette route est accessible seulement à l'administrateur.

- **deleteUserAction($id) :**

Cette méthode est appelée par la route “ `/delete_utilisateur/{id}` ” et permet de supprimer un utilisateur. Cette route est accessible seulement à l’administrateur. Cette fonction vérifie également que l’id en paramètre n’est pas celui de l’administrateur.

6 Résumé des fonctionnalités

6.1 Fonctionnalités implémentées

6.1.1 Fonctionnalités de base

- Espace de connexion.
- Espace d’inscription.
- Gestion des droits (accès à chaque page/fonction contrôlé).
- Création de quiz.
- Répondre à un quiz.
- Espace administration de quiz (ajout, modification, suppression).
- Espace administration de users (suppression).
- Possibilité de mettre du SVG, MathML, texte gras/italique.
- Partage de quiz via l’URL.
- Interface d’installation de l’application (création de la base de données et insertion de quelques données afin de remplir le site).
- Contrôle de quelques données de saisies.

6.1.2 Fonctionnalités bonus

- Bonus 1 : affichage du résultat avec le score et la liste des questions avec la bonne réponse et celle de l’utilisateur.
- Bonus 2 : à la création d’un quiz, l’utilisateur peut choisir le type d’affichage final du résultat.
- Bonus 3 : l’application est responsive et s’adapte pour les terminaux mobiles.

6.2 Fonctionnalités non implémentées

6.2.1 : Fonctionnalités de base

- Vérifier la bonne formation du code SVG et MathML dans les saisies de question et réponses (à l'aide d'XML et d'un parcours DOM).
- Autoriser certaines balises HTML et en interdire d'autres.
- Proposer une interface "dégradée" si le Javascript n'est pas présent sur le navigateur de l'utilisateur.
- Se protéger contre les failles XSS, les Injections SQL ou tout autre type d'attaque.
- Adapter le SVG et le MathML pour qu'il soit responsive.
- Tester que chaque fonction PHP utilisée est disponible au déploiement de l'application. Si une fonction n'est pas disponible, soit on prévient l'utilisateur et on arrête le déploiement, soit on propose une autre solution.

6.2.2 Fonctionnalités bonus

- Bonus 4 : l'auteur d'un quiz peut obtenir des statistiques sur ses quizs.