# Rapport Projet web

RBCS

Table des matières

[Rapport 2](#_Toc9252338)

[1.1 Présentation du site 2](#_Toc9252339)

[1.2 Le choix des technologies 2](#_Toc9252340)

[1.3 Description et utilisation du site 3](#_Toc9252341)

[1.3.1 Les cas d’utilisation du site 3](#_Toc9252342)

[1.3.2 Schéma de navigation 4](#_Toc9252343)

[1.3.3 Organisation de la Base de données 5](#_Toc9252344)

[1.3.4 Gestion du site 6](#_Toc9252345)

[2 Avancement au jour le jour 7](#_Toc9252346)

[2.1 Taches fait par Nicolas 7](#_Toc9252347)

[2.1.1 Entrée sur le site : 7](#_Toc9252348)

[2.1.2 Ajout et modification des informations de l’utilisateur 8](#_Toc9252349)

[2.1.3 Déposer une offre 12](#_Toc9252350)

[2.1.4 Postuler à une offre 14](#_Toc9252351)

[2.1.5 Lister les candidats 15](#_Toc9252352)

[2.2 Taches fait par Hugo 16](#_Toc9252353)

[2.2.1 Page créateur de questionnaires 16](#_Toc9252354)

[2.2.2 Page de réponse à ces questionnaires 19](#_Toc9252355)

[2.2.3 Consultation des résultats 19](#_Toc9252356)

[2.2.4 Page récapitulative de toutes les candidatures d’un candidat 20](#_Toc9252357)

## 1.1 Présentation du site

Notre site, « recruit best computer scientist » (RBCS) permet de mettre en relation des recruteurs avec des personnes en recherche d’emplois. La particularité de notre plateforme est de pouvoir évaluer les compétences des candidats via des questionnaires créés par le recteur lors de la création de son offre d’embauche.

Cela permet de simplifier le processus de recrutement, en faisant une première sélection sur les compétences techniques nécessaire au poste.

Nous avons fait ce choix car nous avons constatés que ça n’existe pas ou en tout pas comme on le voulais.

## Le choix des technologies

Afin de réduire le chargement des données l’architecture du site a été réalisées en MVC2 (Modèle ; Vues ; Contrôleurs ; Contrôleur frontale). Le MVC2 Permet d’avoir un contrôleur frontal qui gère les différentes redirections sur le site. Le premier contrôle se passe au niveau de l’index qui va aller chercher le contrôleur demandé ou la page de connexion si la SESSION n’existe pas encore ou à été détruite. Le contrôleur appelé par le contrôle frontal va gérer l’automatisation de la page ainsi que le traitement des données grâce aux appels au modèle qui contient les requêtes SQL. Une fois les traitements nécessaires effectués le contrôleur chargera la vue adéquate que l’utilisateur pourra visualiser. Grâce à ce concept d’architecture les données générales sont chargés dans l’index et retransmissent aux différents contrôleurs qui vont eu charger les données relatives à la gestion de leurs pages. Les données n’ont donc pas besoin d’être dupliquées ou rechargées.

Les requêtes sont des requêtes préparées afin d’éviter l’injection SQL. PDO prévois des méthodes toutes faite pour l’utilisation des requêtes préparées. L’utilisation est simple mais permet une certaine sécurité.

Nous avons respectivement utilisé un serveur WAMP et LAMP pour réaliser notre développement mais nous utilisons une base de données en ligne afin de pouvoir mettre en commun l’évolution de la base de données ainsi que les données enregistrées.

L’HTML nous a permis de gérer l’interface de visualisation entre l’utilisateur et le serveur ; Le PHP permettait de traiter les données du serveur ; Le CSS et les bibliothèques qui y sont liées tel que Bootstrap nous ont permis de styliser le site ; Le JavaScript et ses bibliothèques tel que JQuery nous ont permis de gérer les pages du côté client. Nous avons aussi utilisé l’AJAX (JavaScript et XML Géré de façon asynchrone) lorsque nous voulons interagir avec le serveur sans que l’utilisateur ne s’en rende compte.

Nous avons utilisé GitHub via l’IDE JetBrains PHPStorm ainsi que le client GitKraken pour gérer l’historique des modifications de manière plus intuitive.

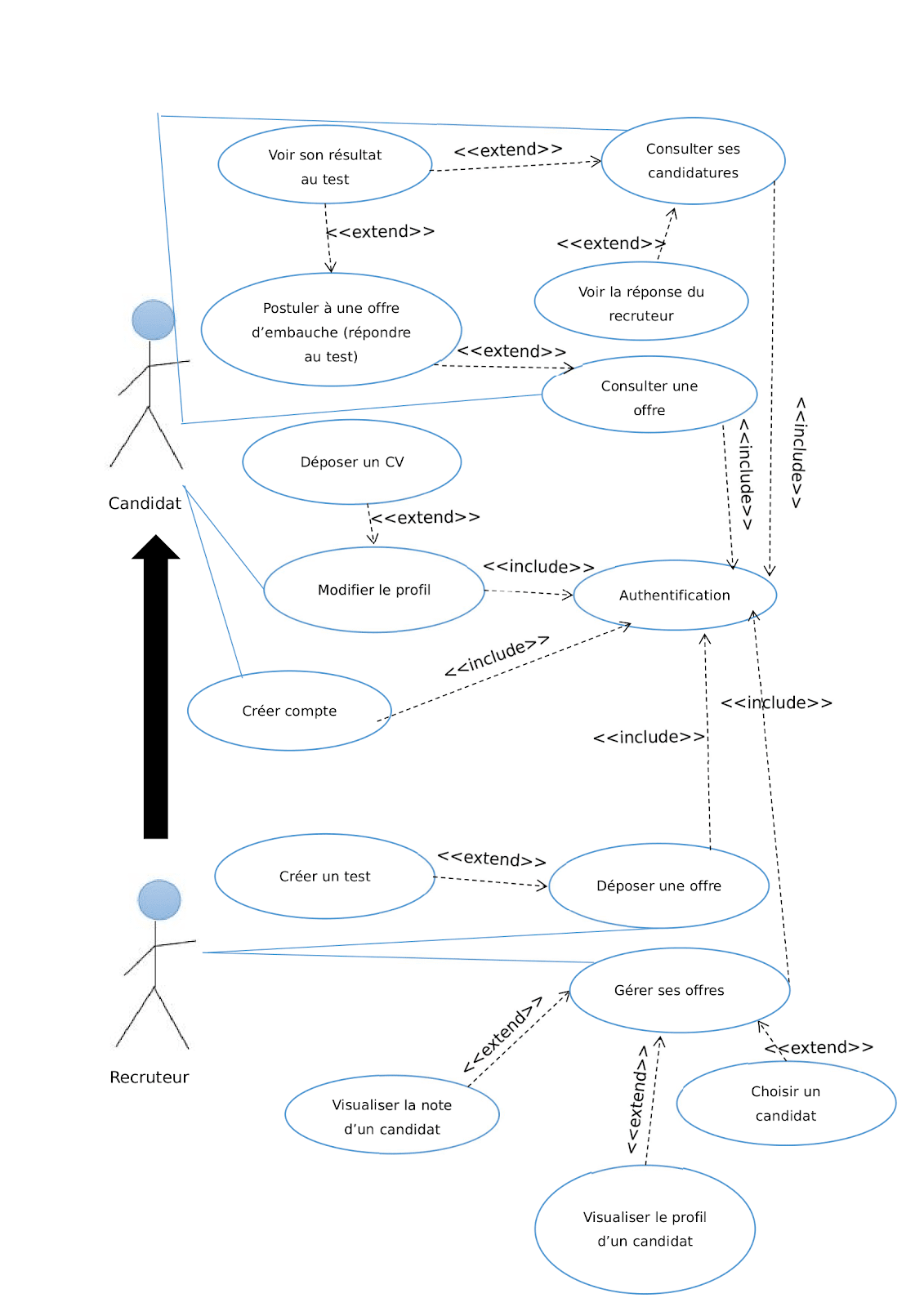
Pour finir afin de rendre le site accessible et le traitement plus rapide nous avons centralisé le site et sa base de données sur l’hébergeur alwaysdata de façon gratuite.

## Description et utilisation du site

### 1.3.1 Les cas d’utilisation du site

Le Diagramme ci-dessous présentes les différents cas d’utilisation de notre site en fonction du type d’utilisateur. La flèche « <<extend>> » signifie que depuis l’action ciblé on peut faire l’action à la racine de la flèche. La flèche « <<include >> » signifie que l’action ciblé est un prés-requis à l’action à la racine de la flèche.

Enfin on notera que le Recruteur est un candidat particulier avec plus d’action. Le recruteur peut faire toutes les actions qui lui sont directement liées, plus celle d’un candidat classique. Un candidat ne peut faire que les actions qui lui directement liées.



### 1.3.2 Schéma de navigation

Page d’accueil

Action : Connexion / inscription

Si première connexion

Page Liste de offres d’embauche

Action : Accéder à l’offre complète

Page d’information utilisateur

Action : Saisie / Mise à jour

Menu

Page Liste de mes candidatures

Action : Accéder Page de correction complète du questionnaire /Accéder à l’offre complète

Page offre d’embauche

Action : Répondre au questionnaire

Page questionnaire

Action : Répondre

Page Correction questionnaire

Redirection

Nouvel onglet

Page Liste de mes offres d’embauche

Action : Accéder Page de Liste des candidatures pour une offre /Accéder à l’offre complète

Page Création d’offre d’embauche

Action : Créer un questionnaire

Page Création de questionnaire

Action : Enregistrer le questionnaire

Redirection

Page Liste des candidats pour une offre

Action : Accéder Page de correction complète du questionnaire pour un candidat / Accéder au profil candidat

Partie recruteur :

Partie candidat :

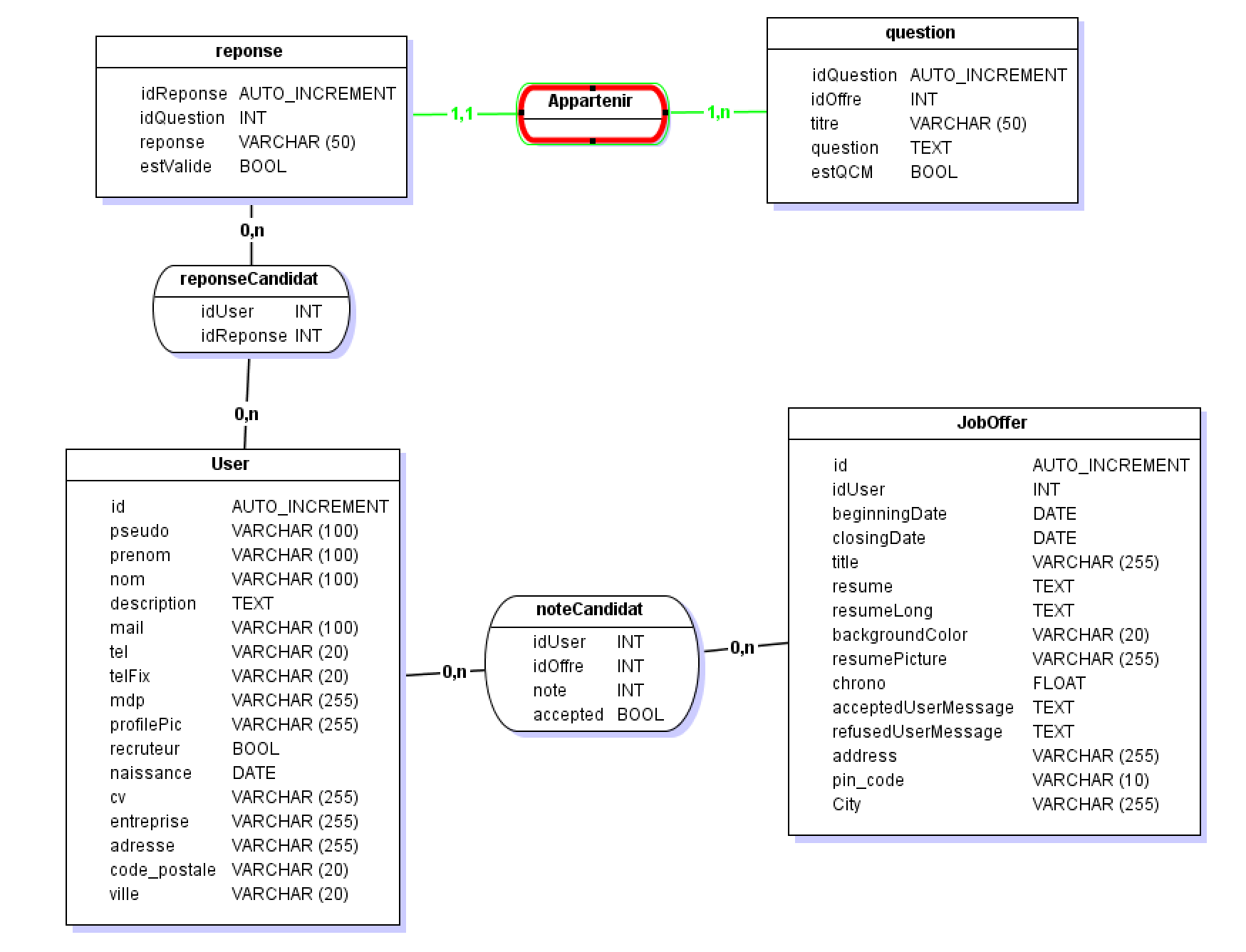
Le Menu n’est pas ne page, il est disponible sur toutes pages qui lui sont liées sur le schéma

Page Profil candidat

Action : télécharger cv

### 1.3.3 Organisation de la Base de données

Pour répondre au besoin du site la base de données est organiser comme suite :



### 1.3.4 Gestion du site

**Compte GMAIL :**

User=>polytechdevweb@gmail.com

password=>dev91Web91Polytech

**Compte Always Data :**

User=>polytechdevweb@gmail.com

password=>dev91Web91Polytech

**Accès FTP :**

hote=>ftp-NicolasEtHugo.alwaysdata.net

port=>21

User=>nicolasethugo

password=>dev91Web91Polytech

**Accès MySQL :**

Serveur=>mysql-NicolasEtHugo.alwaysdata.net

Port=>3306

User=>184058

Password=>dev91Web91Polytech

DataBase=>nicolasethugo\_rbcs

**url pour accéder au site :**

nicolasethugo.alwaysdata.net/RBCS

# Avancement au jour le jour

## 2.1 Taches fait par Nicolas

### 2.1.1 Entrée sur le site :

Pour la connexion je renvoie les erreurs de connexion à partir d’un simple “alert” de JavaScript. Ce n’est pas propre car son affichage dépendra du navigateur mais c’est la seule vue qui est gérée de la sorte.

La fonction TRIM de SQL permet dans les requêtes suivantes d’éviter que l’utilisateur insère des espaces avant ou après les données sans le faire exprès. C’est souvent le cas lorsque l’on fait des “copié/collé”.

#### Sign up :

4 champs sont à remplir : le pseudo, l’adresse de courriel et le mot de passe avec sa confirmation.

Le formulaire est envoyé sur la même page. Grâce au concept du MVC2 l’information est traitée dans le contrôleur. Si les deux mots de passe sont identiques et que le pseudo et courriel de l’utilisateur n’appartiennent pas à un utilisateur différent la fonction sera coupé et une page de renseignements complémentaires sera à remplir.

Dans le cas échéant 2 erreurs peuvent être renvoyées : La confirmation du mot de passe est différente du mot de passe de base ; Le compte utilisateur existe déjà.

Afin de vérifier que l’utilisateur n’existe pas déjà je passe à ma requête deux paramètre : l’adresse de courriel et le pseudo.

*SELECT count(\*) as bool FROM User WHERE  pseudo like TRIM(:login) or mail like TRIM(:login)  or mail like TRIM(:mail)*

#### Sign in :

L’utilisateur doit remplir deux champs soit son identifiant de connexion et son mot de passe. Pour l’identifiant de connexion il pourra utiliser son pseudo ou son adresse de courriel.

La première vérification effectuée prendra en compte les deux champs :

*SELECT id FROM User WHERE  mdp like TRIM(:mdp) AND (pseudo like TRIM(:login) or mail like TRIM(:login))*

Si la vérification précédente échoue nous allons lancer une requête pour savoir si le problème vient de l’identifiant du mot de passe ou que les deux sont totalement erronés.

Savoir si le problème vient de l’identifiant :

*SELECT count(\*) as bool FROM User WHERE  pseudo like TRIM(:login) or mail like TRIM(:login)*

Savoir si le problème vient du mot de passe :

*SELECT  count(\*) as bool FROM User WHERE mdp like :mdp*

Une erreur est affichée selon les 3 cas précédents.

Dans le cas où l’utilisateur serait correctement authentifié il est redirigé vers la liste des offres d’embauche.

Pour chaque fonction du modèle des try/catch sont utiliser pour que je puisse “catcher”/attraper l’erreur et la régler lorsque je travaille en développement.

Je place aussi une condition sur l’exécution des requêtes préparées car la fonction d’exécution retourne “true” si elle s’exécute correctement. Si c’est le cas je vérifie que le nombre de ligne retournée par la requête est cohérente puis que je retourne un booléen afin d’effectuer les comparaisons dans le contrôleur.

*if ($query->execute()){*

*$fetch = $query->fetch(PDO::FETCH\_OBJ);*

*if($fetch->bool == 1){*

*$bool = true;*

*}}*

Étant donné qu’on ne peut pas vérifier l’affichage direct d’une requête préparée il est possible de contourner le problème si nous n’arrivons pas à “catcher” l’erreur mais que la requête ne s’exécute pas. La fonction suivante permet d'afficher l'état de la requête.

*$query->debugDumpParams();*

Nous aurions voulu rajouter en supplément un envoie de courriel automatique pour envoyer le mot de passe à l’utilisateur en cas d'oubli mais nous n’avons pas eu le temps de l’implémenter. Plusieurs bibliothèques comme PHPMailer permettent d’implémenter rapidement cette fonctionnalité.

### 2.1.2 Ajout et modification des informations de l’utilisateur

Je me sers de la même page pour enregistrer les informations relatives au candidat après qu'il se soit inscrit et lorsqu’il veut les modifier.

Le but étant de limiter le travail de l'utilisateur et éviter qu'il ne modifie les informations déjà enregistrées par inadvertance je n’ai qu’à afficher les données de la base si elles existent. Les problèmes se trouvent plus au niveau des pièces jointes comme la photo de profil et le CV. Je suis obligé de modifier les boutons car en cas de modification je dois d’abord écraser les anciens documents du serveur. Je ne peux pas me baser sur le nom du document car j’accepte que plusieurs utilisateurs mettent le même nom à leurs documents.

*//VERIF SI FICHIER EXISTE SI OUI IL FAUT LE RENOMMER*

*if (file\_exists("profiles\_img/".$ppName)){*

*$nomPp = "profiles\_img/".$ppName.rand(0,100000).".".$extension;*

*while (file\_exists($nomPp)){ $nomPp = "profiles\_img/".$ppName.rand(0,100000).".".$extension;  }*

*}*

*Else { $nomPp = "profiles\_img/".$ppName; }*

Les pièces jointes me posent aussi quelques autres problèmes lors de leur enregistrement. Je commence par vérifier que les caractères soient acceptés par le serveur et je les modifie dans le cas contraire.

*$unwanted\_array = array(    'Š'=>'S', 'š'=>'s', 'Ž'=>'Z', 'ž'=>'z', 'À'=>'A', 'Á'=>'A', 'Â'=>'A', 'Ã'=>'A', 'Ä'=>'A', 'Å'=>'A', 'Æ'=>'A', 'Ç'=>'C', 'È'=>'E', 'É'=>'E',*

*'Ê'=>'E', 'Ë'=>'E', 'Ì'=>'I', 'Í'=>'I', 'Î'=>'I', 'Ï'=>'I', 'Ñ'=>'N', 'Ò'=>'O', 'Ó'=>'O', 'Ô'=>'O', 'Õ'=>'O', 'Ö'=>'O', 'Ø'=>'O', 'Ù'=>'U',*

*'Ú'=>'U', 'Û'=>'U', 'Ü'=>'U', 'Ý'=>'Y', 'Þ'=>'B', 'ß'=>'Ss', 'à'=>'a', 'á'=>'a', 'â'=>'a', 'ã'=>'a', 'ä'=>'a', 'å'=>'a', 'æ'=>'a', 'ç'=>'c',*

*'è'=>'e', 'é'=>'e', 'ê'=>'e', 'ë'=>'e', 'ì'=>'i', 'í'=>'i', 'î'=>'i', 'ï'=>'i', 'ð'=>'o', 'ñ'=>'n', 'ò'=>'o', 'ó'=>'o', 'ô'=>'o', 'õ'=>'o',*

*'ö'=>'o', 'ø'=>'o', 'ù'=>'u', 'ú'=>'u', 'û'=>'u', 'ý'=>'y', 'þ'=>'b', 'ÿ'=>'y' );*

*//IMAGE FICHIER*

*$ppName = strtr( addcslashes (  $\_FILES['profilePicNew']['name'] , "'" ) , $unwanted\_array );*

Puis je dois vérifier le type afin que cela ne pose pas de soucis par la suite pour le téléchargement au l’affichage des pièces.

Je commence par créer un tableau avec les formats de fichiers acceptés :

*$extensions\_valides = array('jpg' , 'jpeg' , 'gif' , 'png' , 'ico' , 'svg');*

Je récupère l’extension du fichier :

*$extension = strtolower(  substr( strrchr($ppName, '.')  ,1) );*

Puis je vérifie que l’extension soit bonne :

*if ( in\_array($extension,$extensions\_valides) )*

Je pars du principe que l’utilisateur est responsable des données qu’il enregistre. Un simple message permet de l’avertir que c’est à lui de faire attention aux données qu’il saisit. Cependant je vérifie les données qui pourraient endommager le serveur ou créer des erreurs dans les fonctionnalités du programme.

Mise à part la vérification des pièces jointes le système vérifie aussi que ni le pseudo ni le courriel ne sont déjà utilisé grâce à une fonction similaire à celle de la création de compte. Si cette erreur n’était pas relevée il serait possible qu’il y a une erreur lors de la prochaine connexion.

Pour afficher les erreurs je me sers de simples variables booléennes. Au début je voulais afficher les erreurs en appelant une fonction javascript qui changerait le contenu d’un conteneur de la vue en affichant le message d’erreur au moment où je détectais l’erreur. Néanmoins la fonction n’arrivait pas à trouver le conteneur. En effet la vue n’est créée qu’une fois la vérification effectuée.  J’ai donc décidé de conserver la réponse des vérifications dans une variable booléenne. Une fois le chargement de la vue effectué le système vérifie les variables de chaque erreur et si l’erreur est présente elle est affichée.

*include 'vues/vueSignUpForm.php';*

*echo "</div>";*

*include 'vues/vueBottomMenu.php';*

*include 'vues/vueLateralMenu.php';*

*//Controles*

*if ($pjCv == false){*

*echo "<script>*

*document.getElementById('alertPj').innerHTML =*

*'<div class=\"alert alert-danger\" role=\"alert\">'*

*+'<h4 class=\"alert-heading\">Curriculum Vitae</h4>'*

*+'<p>Le format de votre curriculum vitae n\'est pas accepté.</p>'*

*+'<hr>'*

*+'<p>Les formats acceptés sont : Pdf ; Doc; Docx ; Odt.</p>'*

*+'</div>';*

*</script>";*

Un autre champ qui m’as quelque peu déstabilisé est la “checkbox” qui permet de savoir si l’utilisateur est un recruteur ou un postulant. Au niveau de la récupération elle est soit vide soit avec la valeur 1 mais pour la base de données il lui faut un 0 ou un 1. Je passe donc par une fonction ternaire pour récupérer la valeur que je souhaite.

*$recruteur = ($\_POST['recruteur'] == 1) ? 0 : 1;*

Mais ce n’est pas le plus gros des problèmes. Afin d’assurer une portabilité maximum je tiens à la “responsivité” des pages. Un design responsif est un design qui s’adapte à la taille des fenêtres. Or la “checkbox” était assez large car j’en avait fait un “switch” pour bien faire la différence entre recruteur et candidat. Pour le “design”/style je fonctionne avec des conteneurs qui formes des lignes et des colonnes. Le conteneur en forme de ligne qui comprend la “checkbox” comprends aussi la zone d’affichage de la photo de profile. Lorsque l’espace se fait trop juste je transforme ce conteneur ligne en conteneur colonne afin que les deux éléments ne deviennent pas inutilisables. Un élément css existe pour cela, les “media queries” qui permettent d’adopter un certain style en fonction de la taille de la fenêtre.

*@media (min-width: 1176px) {*

*.divColAdaptative{*

*display: flex;*

*flex-direction: row;*

*justify-content: center; }*

*#conteneurImg{ ... }*

*}*

*@media (max-width: 1175px) {*

*.divColAdaptative{*

*display: flex;*

*flex-direction: column;*

*justify-content: center;*

*}*

*#conteneurImg{*

*...*

*}*

*}*

Un petit plus pour que l’utilisateur puisse voir directement la photo qu’il exporte sur le serveur ou pré remplir le nom du CV au cas où il ne voudrais pas le modifier à été réalisable grâce à JavaScript.

Pour l’affichage de la photo j’ai été chercher du code sur internet qui permet de lire le fichier puis le placer dans le conteneur adéquat.

*//APERCU FICHIER*

*function readURL(input) {*

*if (input.files && input.files[0]) {*

*var reader = new FileReader();*

*reader.onload = function (e) {*

*$('#blah')*

*.attr('src', e.target.result)*

*.attr('alt', 'aperçu de ce fichier non disponible');*

*};*

*reader.readAsDataURL(input.files[0]);*

*}*

*}*

Tandis que pour le cv j’avais trouvé une fonction qui permettait d’écouter le bouton qui permet l’import du fichier sur le serveur.

*var inputPjCv = document.getElementById("pjCv");*

*inputPjCv.addEventListener("change", handleFiles, false);*

*function handleFiles() {*

*var file = this.files;*

*document.getElementById('cv').value = file[0].name*

*}*

Une fois qu’on a récupéré le tableau de fichiers chargés il est possible de récupérer son nom ou encore sa taille puis d’en faire ce que l’on veut.

Le problème est que je demande au système de remplacer mon ancien bouton d’upload lorsqu’un CV existe déjà pour éviter de surcharger le serveur comme je l’expliquait précédemment. D’autres moyens auraient été possible pour éviter ce souci.

Cependant j’ai choisi d’utiliser cette technique et la fonction précédente renvoie une erreur si le bouton est désactivé. En effet un écouteur écoute l’élément constamment, il ne peut donc pas être caché. L’autre moyen que j’ai trouvé est d’utiliser l’écouteur HTML “onchange” qui appelle la fonction qu’au moment où elle est nécessaire, comme pour la photo.

*function autoNameCv(input){*

*var file = input.files;*

*document.getElementById('cvNew').value = file[0].name*

*}*

### 2.1.3 Déposer une offre

Pour créer une offre d’embauche la seule condition préalable est d’être un recruteur. Par la suite les traitements ne sont pas bien différents du formulaire qui permet d’enregistrer des informations sur les utilisateurs.

Grâce au concept du MVC2 Je n’ai qu’à regarder les données concernant l’utilisateur que j’utilise par la suite dans le menu latéral pour bloquer l’accès aux simple candidats à certaines fonctionnalités. Pour les candidats plus futés qui essaient de passer directement par les URL un deuxième contrôle est effectué sur la vue. Aucune action n’est effectuée par le système mais un petit message permet de prévenir l’utilisateur de son action malveillante et de la dissuader d’insister.

*elseif ($user['recruteur'] != 1){*

*echo "*

*<div class=\"rowPerso\">*

*<div class=\"alert alert-danger\" role=\"alert\">*

*<h4 class=\"alert-heading\">Hum Hum !</h4>*

*<p>Vous n'êtes pas autorisé à utiliser cette fonctionnalité.</p>*

*<hr>*

*<p class=\"mb-0\">Nous nous réservons le droit de vous bannir en cas de récidive.</p>*

*</div></div>";*

*}*

Un nouveau type de vérification est effectué par rapport aux formulaires des informations personnelles. Nous n’avons pas encore eu le temps de l’implémenter mais le formulaire qui permet de tester les utilisateurs est normalement aussi soumis à une durée limitée. Un “settimeout” aurait pu permettre de soumettre le formulaire en fonction du temps donné dans un premier temps. Afin de prévenir d’une future implémentation ou de l’enregistrement dans la base de données le système vérifie que le temps soit un entier positif ne commençant pas par un 0. Je réalise cette vérification grâce à une expression régulière. Si la condition n’est pas respecté l’utilisateur est informé de son erreur.

*preg\_match("/^[1-9][0-9]\*$/",  $\_POST['chrono'])*

Un autre champ qui est géré mais dont l’importance est moindre est le champs de date de l’offre. Grâce à la librairie “datepicker” de JQuery il est possible de paramétrer assez facilement les champs pour que la date de début ne soit pas inférieure au jour actuel et que la date de fin soit supérieure à la date de début :

*$(document).ready(function() {*

*$( "#datepickerDebut" ).datepicker({*

*dateFormat: 'dd/mm/yy',*

*defaultDate: "+0w",*

*numberOfMonths: 2,*

*changeMonth: true,*

*changeYear: true,*

*yearRange: '-100:+5',*

*minDate:new Date(),*

*//maxDate: '+1Y',*

*onClose: function( selectedDate ) {*

*$( "#datepickerFin" ).datepicker( "option", "minDate", selectedDate );*

*}*

*});*

*$( "#datepickerFin" ).datepicker({*

*dateFormat: 'dd/mm/yy',*

*defaultDate: "+0w",*

*numberOfMonths: 2,*

*changeMonth: true,*

*changeYear: true,*

*//maxDate: '+2Y',*

*onClose: function( selectedDate ) {*

*$( "#datepickerDebut" ).datepicker( "option", "maxDate", selectedDate );*

*}*

*});*

*});*

Si la photo du formulaire est valide et que le chrono est au bon format les données du formulaire seront insérées et je récupère l’identifiant du formulaire grâce à une fonction PDO :

*$insert = $bdd->lastInsertId();*

Mon contrôleur ajoutera cette identifiant à une variable. Lorsque celui-ci est détecté par le vue cette dernière affiche un bouton pour créer le formulaire de test des candidats. Le recruteur n’est pas obligé de le créer mais c’est son seul moyen d’avoir des candidats pour son offre sur la plateforme.

Si le bouton “Oui” est cliqué la vue de création de test est appelé avec l’identifiant de l’offre en paramètre sinon le recruteur est redirigé vers l’historique des offres qu’il a posté.

Un tableau existe pour récapituler les offres du recruteur ce dernier permet de voir l’offre ainsi que d’accéder aux tableaux des candidats dont nous parlerons plus tard. Un bouton d’édition et de suppression de l’offre sont aussi présents mais leurs fonctionnalités n’ont pas été implémentés par manque de temps.

2.1.4 Postuler à une offre

Tous les utilisateurs peuvent participer à une offre. La seule condition est que l’offre ne soit pas dépassée, qu’elle possède un formulaire de test et que l’utilisateur n’y ait pas déjà participé. Pour afficher l’ensemble des offres aucun système de tri n’a été implémentés mais le système va chercher seulement les offres dont la date de fin est supérieur ou égale à la date du jour.

*SELECT \*, JobOffer.id as idOffre, DATE\_FORMAT(beginningDate, '%d/%m/%Y') as dateD, DATE\_FORMAT(closingDate, '%d/%m/%Y') as dateF FROM JobOffer INNER JOIN User ON JobOffer.idUser = User.id WHERE closingDate >= '".date("Y-m-d")."*

Nous pouvons aussi constater des fonctions de traitement des dates dans la fonction. En effet au lieu de devoir modifier l’affichage des dates en php et de les modifier à nouveau pour les insertions j’ai préféré traiter les dates directement à l’aide des fonctions SQL.

Insertion :

*STR\_TO\_DATE(:naissance, '%d/%m/%Y')*

Sélection :

*DATE\_FORMAT(closingDate, '%d/%m/%Y') as dateF*

Lorsque l’utilisateur arrive sur l’offre il a accès à toutes les informations nécessaires pour envisager son futur poste et contacter le recruteur. En descendants il pourra cliquer sur un bouton lui permettant de réaliser le test s’il existe. L’utilisateur verra un message d’erreur le cas échéant.

*if ($testExist){*

*?>*

*<button type="submit" class="btn btn-success" onclick="window.location.href = '?p=ctrlTest&offerId=<?php echo $offre['id'] ?>'">Participer au test</button>*

*<?php*

*}*

*else{*

*?>*

*<h3 style="color: red;text-align: center">Aucun test n'existe pour cette offre</h3>*

*<?php } ?>*

Si l’utilisateur postule pour la première fois il sera redirigé vers le test qui sera ensuite enregistré dans son historique et dans le tableau des candidats accessible par le recruteur via l’historique de ses offres.

Dans le cas où l’utilisateur aurait déjà postulé il sera automatiquement redirigé vers son résultat au test.

*$participationEffectuee = alreadyParticipate($bdd, $offerId, $idUser);*

*$url = ($participationEffectuee) ? "vues/form\_generate/ConsulterResultatQC.php?idOffre=".$offerId."&idCandidat=".$idUser."" : "vues/form\_generate/GenFormQCM.php?idOffre=".$offerId."&idCandidat=".$idUser."" ;*

*echo "<script>window.location.href='".$url."'</script>";*

### 2.1.5 Lister les candidats

Comme pour créer une offre d’embauche seule le recruteur à accès à l’historique de ses offres d’embauches et à la liste des candidats qui en découle. De cette liste il peut cliquer sur le candidat et voir son profil, observer le résultat de son test puis l’accepter ou le refuser. Lorsqu'il consulte le profil de l’utilisateur j’ai décidé de rajouter l’attribut “target=’\_blank’” à la balise du téléchargement du CV afin qu’il ne quitte pas l’onglet en cours d’utilisation.

Lorsqu’il a rempli l’offre d’embauche le recruteur a saisi un message automatique pour les utilisateurs acceptés ou refusés.

Lorsqu’il valide ou refuse un candidat ce dernier pourra voir le message qui y est lié dans son historiques de candidatures. Pour le recruteur il sera simplement averti que le traitement de la candidature à bien été effectué et ne pourras plus revenir sur ça décision. Grâce au concept du MVC2 il est assez simple de traiter les données sur la même page et de mettre à jour la ligne de l’utilisateur pour que le recruteur ne puisse plus la modifier.

*$update = false;*

*if(isset($\_GET['accepted'])) {*

*$update = answeredToUser($bdd, $offerId, $\_GET['idCandidat'], $\_GET['accepted']);*

*}*

*$lesCandidats = listeDesCandidats($bdd, $offerId);*

*include 'vues/vueTopMenu2.php';*

*echo "<div style='padding-top: 10vh;padding-bottom: 10vh'>";*

*include "vues/vueApplicants.php";*

*echo "</div>";*

*include 'vues/vueBottomMenu.php';*

*include 'vues/vueLateralMenu.php';*

*if($update){*

*echo "<script>*

*document.getElementById('choixUtilisateur').innerHTML =*

*'<div class=\"alert alert-success\" role=\"alert\">'*

*+'<h4 class=\"alert-heading\">Réponse envoyée</h4>'*

*+'<p>L\'utilisateur vient d\'être averti de votre réponse.</p>'*

*+'</div>';*

*</script>";*

*}*

# 2.2 Taches fait par Hugo

Ma partie était de réaliser les pages suivantes :

* Une page servant à créer des questionnaires, comportant plusieurs questions.
* Une page pour répondre à ces questionnaires
* Une page pour consulter le résultat / note d’un questionnaire
* Une page regroupant l’ensemble des candidatures faite par un candidat (celui connecter)

### Page créateur de questionnaires

Le créateur de questionnaires est une page accessible pour les utilisateurs du type recruteur.

Elle leur permet d’ajouté un ensemble de question à leur offre d’embauche. Ces questions sont soit des QCM (question à choix multiple) soit des QCU(question à choix unique), donc soit checkbox ou soit radio en HTML.

Découpage en taches :

* Réalisation du formulaire servant à créer une question.
* Réalisation d’une page avec un drag and drop comportant deux zones :

1er pour attraper la question à ajouter (QCM ou QCU)

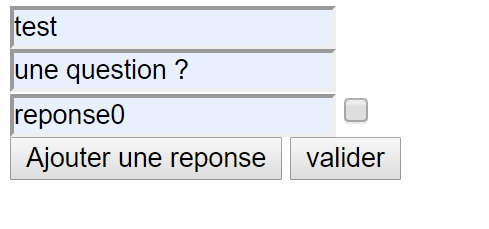
2ème qui contient toutes les questions

* Au moment du drop dans la 2ème zone ajouter un nouveau formulaire de création de question à la page.
* Traitement php : insertions dans la BD de l’ensemble des questions et réponses associées

#### Réalisation du formulaire servant à créer une question

Formulaire simple fait en HTML, comparant les champs : Titre, Question et un nombre variable de réponse.

La capture ci-dessous montre le formulaire ce création d’une question :

L’objectif est donc de pouvoir ajouter les réponses une à une en appuyant sur « Ajouter une réponse ». On pourra indiquer si la réponse fait partie des bonnes réponses par la checkbox a sa droite.

La première difficulté à donc était de pouvoir avoir un nombre de réponse variable.

Première solution : à chaque appuie sur le bouton on appelle une fonction JavaScript qui ajoute dans la « div n » la « réponse n » et la « div n+1 » et incrémente n. Problème de structure : toutes les réponses sont imbriquées les unes dans les autres. *On rappellera que mes connaissances en JS à ce moment sont inexistantes.*

Deuxième solution *: Suite à quelque recherche et l’aide de Nicolas je découvre « element.innerHTML ».* Fonction JavaScript permettant de récupère le contenu HTML d’un élément. A l’aide du « innerHTML » on obtient une meilleure solution : on peut récupère le contenue de la « div réponse », y ajouter la nouvelle réponse et mettre à jour la « div réponse ». soit :

*element.innerHTML  = element.innerHTML + nouvelleQuestion ;*

Avec cette solution on supprime le problème d’imbrication de la solution précédente. Mais, nouveau problème : à chaque ajout d’une nouvelle réponse on perd le contenue des réponses précédentes. En effet, « innerHTML » ne copie que la structure html sans le contenue.

Il faut donc faire autrement…

Solution finale : *Suite à des recherche je découvre « element.*appendChild()*». Avec cette fonction on peut ajouter des éléments HTML à la fin de l’élément sélectionné. C’est exactement ce qu’on, on ajouter une nouvelle réponse à la « div réponse ». Soit :*

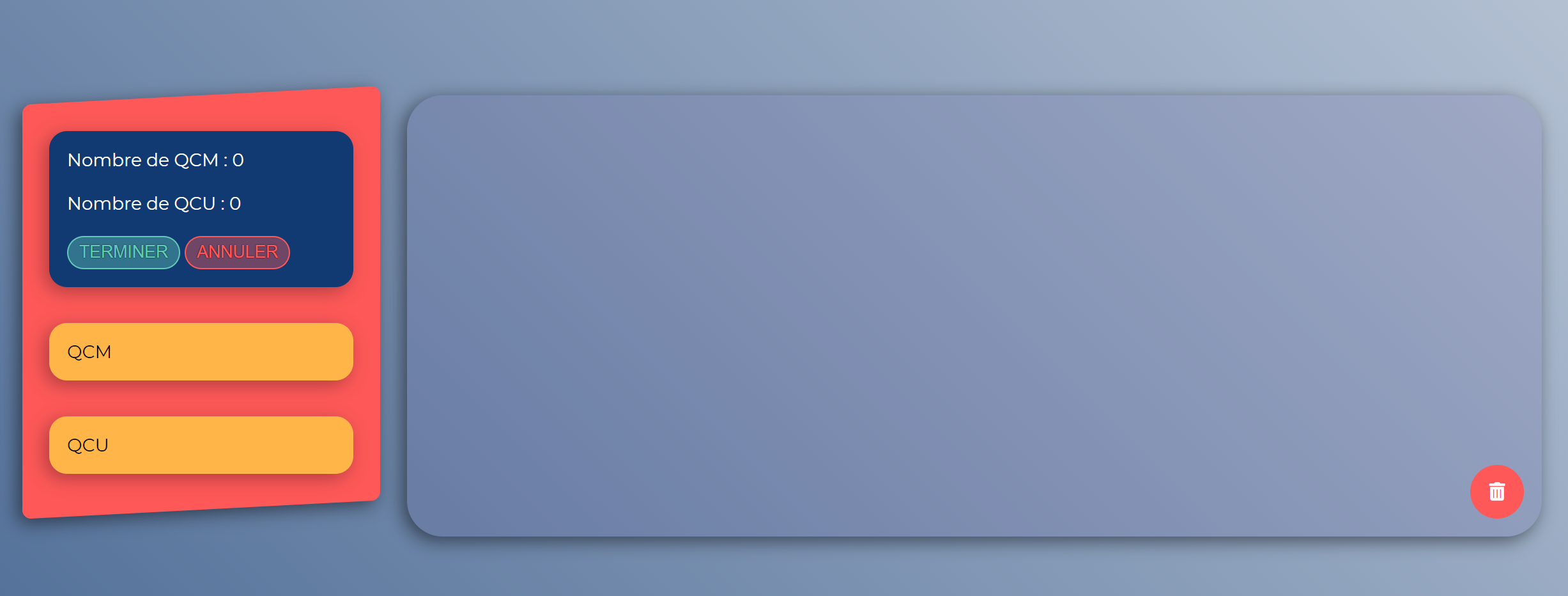
*element.*appendChild(*nouvelleQuestion*) ;

Le formulaire de création d’une question est fonctionnel, reste à le rendre paramétrable pour qu’il s’adapte à la création d’un QCM ou QCU. La solution est triviale : passage en méthode GET du type de question et petit traitement php pour le choix entre checkbox et radio.

#### Réalisation de la page de création de questionnaire avec un drag and drop

Cette page permet de créer un ensemble de question à l’aide d’une interface drag and drop.

Ci-dessous la page de création de questionnaire.



Cette page à était réaliser avec le drag an drop natif à HTML 5. La prise en main cette fonctionnalité n’a pas était simple. Certaine fonctionnalité prévue initialement ont dû être abandonné : Comme le déplacement des questions déjà créer et la suppression de question.

Dans un premier temps l’objectif était de découvrir une manière de faire un drag and drop classique. Soit le déplacement d’élément d’une zone A vers une zone B. ensuite, de l’adapter à notre besoin soit copier l’élément de la zone A et le coller dans la zone B puis modifier l’élément coller dans la zone B pour faire apparaitre le formulaire.

On a en fait pour zone A la zone à droite qui est la partie sélection d’élément à créer. Les éléments de cette zone doivent être attrapable et déplaçable (attribue « draggable = true » en HTML) vers la zone B, celle de droite où l’on va instancier un nouveau formulaire de création de question.

La Difficulté principale à était de transformé l’élément une fois dans la zone de droite pour devenir le formulaire de création de question.

Première solution : mettre à jour le contenue de la div représentant la zone de droite avec « *element.innerHTML » ou « element.*appendChild » en ajoutant le formulaire de création de question. Problème, la partie JavaScript contenue dans la page de création de question n’est plus interpréter.

Deuxième solution : Utilisation d’iFrame, soit charger une ou plusieurs pages complètes dans une autre. Cela fonctionne dans le principe mais à poser encore plus de problème : difficulté du post traitement php car on a plusieurs pages, on ne peut pas changer la taille de l’iFrame une fois créer et c’est vraiment moche.

Solution finale : transformation de la page php qui contient le formulaire de création d’une question en une fonction JavaScript qui vas créer tous les éléments HTML et les insérer dans la bonne zone. Cette fonction JS est directement dans la page principale. Elle est appelée a chaque drop d’un éléments venant de la zone à gauche dans la zone à droite.

Initialement on voulait pouvoir déplacer les questions ente elle pour pouvoir changer l’ordre, on n’a pas réussi. En effet il nous a était impossible de séparer le cas drag and drop de la zone à gauche vers la zone à droite du cas drag and drop de la zone à droite vers la zone à droite. Hors lors du drag and drop on fait des copies de l’élément sélectionné pour l’insérer dans la zone de drop avec un nouvel identifiant. Lors de la copie on perd se qui avait était saisie et on créer des trous dans les identifiants, La boucle du traitement php qui doit insérer l’ensemble des questions / réponses peut s’arrête trop top.

Pour la même raison on n’a pas réussi à supprimer une question une fois créer.

Le traitement post en php permet l’insertion de l’ensembles des questions avec leurs réponse dans la BD. Chacune des questions est liée par un identifiant a l’offre auquel est rattaché le questionnaire.

### Page de réponse à ces questionnaires

Cette page permet à un candidat de répondre à un formulaire. Pour cela la page est constituée de l’ensemble des questions liée à l’offre à laquelle le candidat veut postuler. Ce traitement est fait en php et nécessite deux informations :

* L’identifiant de l’offre pour charger tous les questions et réponses
* L’identifiant du candidat qui y répond pour sauvegarder ses réponses

Les questions sont chargées depuis la BD à l’aide de la requête SQL suivante :

$sql = 'SELECT idQuestion, titre, question, estQCM FROM question WHERE idOffre = '.$\_GET["idOffre"];

Pour chaque questions il faut en plus charger les réponse pour chaque question, traitement fait à l’aide de la commande SQL suivante :

$sql2 = 'SELECT idReponse, idQuestion, reponse FROM reponse WHERE idQuestion = :idQuestion';

Enfin pour insérer les réponses fournies par le candidat, pour cela on enregistre en BD seulement les réponse coché par le candidat à l’aide de la requetés SQL suivante :

$sql = "INSERT into reponseCandidat (idCandidat, idReponse) VALUES ( :idCandidat, :idReponse)";

### Consultation des résultats

Cette page permet de voir le résultat et la note d’un candidat à un questionnaire.

Cette page a donc besoin de deux informations pour fonctionner : identifiant du candidat et identifiant de l’offre d’on on souhaite voir le résultat de questionnaire. Lors du traitement php qui boucle sur l’ensemble des question réponse à afficher on va aussi chercher les réponses données par le candidat pour les comparer aux bonnes réponses.

Pour chaque réponse on change la couleur de la div qui la contient pour plus de lisibilité. Les bonnes réponses cochées par le candidat sont en vert foncé, les mauvaises réponses cochées sont en rouge et les bonnes réponses non coché sont en vert clair.

Ainsi on peut établir une note, mais la manière de compter les points n’est pas simples et il a fallu faire un choix.

Sachant que chaque question doit rapporter 1 point si toutes les bonnes réponses on était donné et qu’il n’y a pas de mauvaise réponse. Le point de la question ne peut pas être négatif.

Premier système de notation : on ajoute 1 / (nombre de réponse) par bonne réponse et on Ajoute – 1 / (nombre de réponse) par mauvaise réponse. Problème si on coche tout on obtient des points.

Deuxième système de notation : on ajoute (1 / nombre de bonne réponse) par bonne réponse et on Ajoute – 1 / (nombre de mauvaise réponse) par mauvaise réponse. Problème s’il y a qu’une mauvaise réponse et quelle est coché la réponse vaut 0.

Troisième système de notation : on ajoute (1 / nombre de bonne réponse) par bonne réponse et on Ajoute – 1 / 2 par mauvaise réponse et 0 si tout est coché. C’est ce système qu’on a choisi.

La note est affichée sur la page et aussi insérer dans la BD si c’est la première fois qu’elle est calculé.

### Page récapitulative de toutes les candidatures d’un candidat

Cette page est un tableau récapitulatif des candidatures du candidat connecter. Il n’y a pas eu de difficulté particulière. Simple traitement php qui boucle sur l’ensemble des candidatures présente dans la BD le candidat connecter. Sur chaque ligne du tableau on trouve le nom de l’offre, la note au questionnaire et la réponse du recruteur s’il l’a donnée.

On peut cliquer sur le titre de l’offre pour y être rediriger, de la même manière on peut cliquer sur la note pour voir la page de résulta complète du test.

Enfin si le recruteur à donner sa réponse on pourra voir un message détaillant la réponse, ce message apparait sous forme de fenêtre modale. Cette fonctionnalité utilise de l’Ajax. Le principe est d’appelé une page php avec des données entres a traité a d’afficher sont retour dans la fenêtre modale.

Ci-dessous, un exemple de la fenêtre modale :

