Validation inscription :

1. Validation des données : Avant de procéder à l'inscription, assurez-vous de valider les données fournies par l'utilisateur, en particulier le pseudo, l'adresse e-mail et le mot de passe. Vérifiez la longueur minimale et maximale, les caractères autorisés, les contraintes spécifiques, etc. Cette validation préliminaire aidera à éviter les problèmes ultérieurs liés aux données incorrectes.
2. Gestion des erreurs : Lorsque vous renvoyez des messages d'erreur dans la fonction **Toolbox::dataJson()**, assurez-vous de ne pas divulguer d'informations sensibles sur les raisons exactes de l'échec de l'inscription. Vous pouvez simplement renvoyer un message générique sans spécifier si le pseudo ou l'e-mail est déjà utilisé.
3. Utilisation de **password\_hash()** : Vous utilisez correctement **password\_hash()** pour hacher le mot de passe avant de l'enregistrer en base de données. Assurez-vous simplement que la colonne de votre base de données a une longueur suffisante pour stocker correctement le hachage du mot de passe.
4. Création du token de validation : Assurez-vous que la fonction **createToken()** génère un token aléatoire et unique pour chaque utilisateur. Vous pouvez utiliser des fonctions de hachage, comme **password\_hash()**, pour générer des tokens sécurisés.
5. Envoi de l'e-mail de confirmation : Vérifiez que la fonction **Toolbox::sendMail()** utilise des pratiques sécurisées pour l'envoi d'e-mails, telles que l'utilisation de serveurs SMTP sécurisés et l'inclusion d'un contenu approprié dans le corps du message pour éviter les attaques XSS.
6. Manipulation des erreurs : En cas d'erreur lors de l'envoi de l'e-mail de confirmation, évitez de divulguer des informations sensibles. Par exemple, au lieu de dire "un problème est survenu lors de l'envoi du mail de confirmation", vous pouvez simplement indiquer "un problème est survenu lors de la finalisation de l'inscription".
7. Logique de succès de l'inscription : Assurez-vous que la logique de réussite de l'inscription (**Toolbox::dataJson(true)**) est exécutée uniquement si l'e-mail de confirmation est envoyé avec succès.

Coté client :

Pour une meilleure expérience utilisateur, vous pouvez envisager d'améliorer votre fonction de vérification en effectuant les actions suivantes :

1. Réinitialisation des erreurs : Lorsque l'utilisateur corrige une erreur, pensez à réinitialiser les styles d'erreur précédemment appliqués aux champs. Par exemple, supprimez les classes **inputError** ou réinitialisez les messages d'erreur pour les labels.
2. Validation en temps réel : Vous pouvez ajouter des événements de saisie ou de perte de focus pour valider les champs à mesure que l'utilisateur les remplit. Cela permet de fournir des commentaires immédiats sur la validité des données, avant même la soumission du formulaire.
3. Validation renforcée : En plus des vérifications de base, vous pouvez également ajouter des validations supplémentaires pour renforcer la sécurité et la qualité des données. Par exemple, vérifiez la force du mot de passe (caractères spéciaux, majuscules, minuscules, chiffres), ou effectuez une vérification côté client pour éviter les doublons de pseudo ou d'adresse e-mail.
4. Gestion des erreurs de manière cohérente : Assurez-vous de gérer les erreurs de manière cohérente à la fois côté client et côté serveur. Utilisez des messages d'erreur clairs et explicites pour guider les utilisateurs lors de la correction des erreurs.

FLAG :

Si vous décidez de créer une table flags pour gérer les signalements de contenu inapproprié dans votre forum de discussion, voici quelques champs qui pourraient être utiles à inclure dans cette table:

* flagId: Un identifiant unique pour chaque signalement, généralement un entier auto-incrémenté.
* topicId: Une clé étrangère qui relie chaque signalement à un sujet dans la table topics.
* userId: Une clé étrangère qui relie chaque signalement à un utilisateur dans la table des utilisateurs (par exemple, users). Ce champ pourrait être utilisé pour stocker l’identifiant de l’utilisateur qui a effectué le signalement.
* flagDate: La date et l’heure du signalement, généralement stockées sous forme de champ de type datetime ou timestamp.
* flagReason: La raison du signalement, généralement stockée sous forme de champ de type text ou varchar. Ce champ pourrait être utilisé pour permettre aux utilisateurs de fournir une explication sur la raison pour laquelle ils ont signalé un sujet comme inapproprié.

Ces champs vous permettraient de stocker des informations détaillées sur les signalements dans votre base de données et de gérer efficacement les signalements dans votre forum. Bien sûr, vous pouvez ajouter ou supprimer des champs en fonction de vos besoins spécifiques. J’espère que cela répond à votre question! 😊

Dans ma base de données de mon site internet (un forum de discussion) j'ai :

Une table user : userID, pseudo, email, userDate (date d'inscription)…

Une table topics : topicID, title\_topic, topicMessage, topicDate, categoryID (clé étrangère de la table categorys), userID (clé étrangère de la table

**Retour en base 64 pour enregistrer les images :**

Enregistrer les images en base64 à plusieurs avantages :

* Pas besoin d’ouvrir une autre connexion et de faire une requête HTTP au serveur pour l’image .

Le script JS communique une fois avec le serveur, enregistre la chaine qui contient l'image en base64 et donne une réponse au script qui peu mettre à jour le DOM

* Très simple à mettre en œuvre, il n'y a quasiment rien à faire

Détail du code et explication :

**Vérification coté client :**

Concernant les messages vide ET les balises <img… qui n'ont pas la chaine data:image . Pourquoi ?

Par ce que on peux choisir d'uploader un pdf par exemple, et la chaine en base 64 sera : <img src="data:application…>et donc ce n'est pas une image.

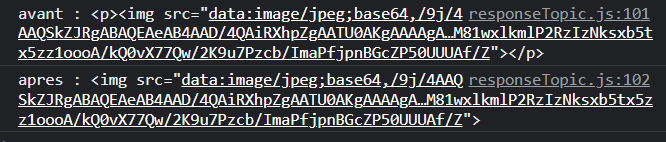
        if (!quill.root.innerHTML.replace(/<[^>]\*>/g, match => match.includes('<img src="data:image/') ? match : '')) {

            throw new Error(`Veuillez entrer du contenu !`);

        }

|  |  |
| --- | --- |
| pour la chaine "bonjour" | Pour la chaine "" |

Pour une image seule :



Si elle n'a pas exactement cette chaine "<img src="data:image/" elle sera rempalacé par ""

En gros ça supprime toutes les balises (pas leurs contenu) sauf si c'est une balise img valide. Ainsi, si le user rentre une chaine, il restera juste la chaine. Si le user met uniquement une ou plusieure images (même sans texte), il restera l'image(s). Par contre le si user n'entre rien, ou si il upload un fichier qui ne ressemble pas à ça : "<img src="data:image/ ", alorson supprime les balises, il reste quoi ? bah rien du tout. Donc : message vide

*//on boucle sur toutes les images pour s'assurer que le message du User ne comporte pas d'image supérieur à 300ko et que le type mime correspond aux contenu du tableau de type autorisé*

*//console.log(array1.includes(2));*

            const TypeMimeAuthorized = [

                "image/jpg",

                "image/jpeg",

                "image/gif",

                "image/png"

            ];

            for (const image of images) {

                const imageBase64 = image.src;

                const blob = await fetch(imageBase64).then(res => res.blob());

                if (blob.size > 307200) {

                    throw new Error(`Le poids de l'image doit être inférieure à 300ko !`);

                }

                if (!TypeMimeAuthorized.includes(blob.type)) {

                    throw new Error(`Le fichier n'est pas une image valide. Extensions autorisées : png, gif ou jpeg(jpg) !`);

                }

            }

*//si type et poids ok, le code continu ...*

**Vérification coté serveur :**

Exemple :

$htmlContent = '<p><img src="data:image/jpeg;base64,/9j/4AAQ...(CODE BASE64 IMAGE1)...PfjpnBGcZP50UUUAf/Z" alt="Z"></p><p>et mon autre image : </p><p><img src="data:image/jpeg;base64,/9j/4AAQSkZJ...(CODE BASE64 IMAGE2)...ZP50UUUAf/Z" alt="Z"></p>';

Ici on a deux images en base 64.   
Objectif : vérifier son type mime et son poids.

preg\_match\_all('/src="data:([^;]+);base64,([^"]+)"/', $htmlContent, $matches);

        $mimeTypes = $matches[1];

        $base64Strings = $matches[2];

Cette expression régulière est utilisée pour extraire à la fois le type MIME et la chaîne base64 des images à partir du contenu HTML. Voici une explication de chaque partie de l’expression régulière :

/: Le délimiteur indiquant le début et la fin de l’expression régulière.

src="data:: Recherche les chaînes qui commencent par src="data:.

([^;]+): Recherche un ou plusieurs caractères qui ne sont pas des points-virgules. Les parenthèses indiquent que nous voulons capturer cette partie de la chaîne correspondante. Cette partie de l’expression régulière capture le type MIME de l’image.

;base64,: Recherche les chaînes qui contiennent ;base64, après le type MIME.

([^"]+): Recherche un ou plusieurs caractères qui ne sont pas des guillemets. Les parenthèses indiquent que nous voulons capturer cette partie de la chaîne correspondante. Cette partie de l’expression régulière capture la chaîne base64 de l’image.

": Recherche les chaînes qui se terminent par un guillemet.

/: Le délimiteur indiquant la fin de l’expression régulière.

Lorsque cette expression régulière est utilisée avec la fonction preg\_match\_all(), elle extrait à la fois le type MIME et la chaîne base64 des images à partir du contenu HTML. Le type MIME est stocké dans le premier élément du tableau de résultats ($matches[1]) et la chaîne base64 est stockée dans le deuxième élément du tableau de résultats ($matches[2]).

Résultat du dump()



Avec une seule boucle on va pouvoir vérifier le type mime des images et leur taille :

        preg\_match\_all('/src="data:([^;]+);base64,([^"]+)"/', $htmlContent, $matches);

        $mimeTypes = $matches[1];

        $TableauBase64 = $matches[2];

*//tableau type mime autorisés*

        $TypeMimeAuthorized = [

            "jpg" => "image/jpg",

            "jpeg" => "image/jpeg",

            "gif" => "image/gif",

            "png" => "image/png"

        ];

        foreach ($TableauBase64 as $index => $base64Chaine) {

            $mimeType = $mimeTypes[$index];

            if (!in\_array($mimeType, $TypeMimeAuthorized)) {

                echo "Le fichier n'est pas une image valide. Extensions autorisées : png, gif ou jpeg(jpg)";

                exit;

*//on stop le code*

            }

            $base64ChaineDecode = base64\_decode($base64Chaine);

            $tailleImage = strlen($base64ChaineDecode);

            if ($tailleImage > 307200) {

                echo "Le poids de l'image doit être inférieure à 300ko";

                exit;

*//on stop le code*

            }

        }

        echo 'ok';

        exit;