BTS SIO
Services Informatiques aux Organisations
LYCÉE BONAPARTE

Rapport d'un BTS SIO

★ *Stage : Evoliz*
Semaine 3

BRUEL LUCAS
27/05/2023

Sommaire

I/ Le stage.

Dans cette partie, je souhaite rappeler toutes les informations concernant mon stage tel que le cadre qui définit mon stage et le but et une présentation de l'entreprise.

A/ Définition du stage.

Mon stage se déroule du 9 Mai au 2 juillet 2023, en tant que développeur dans une entreprise proposant une solution de facturation facile accompagnée par ces partenaires, nommée Evoliz.

Durant ce stage, avec l'aide de mon tuteur de stage Monsieur Julien Libert, je travaillerais dans le développement, la maintenance, l'amélioration, l'ajout de fonctionnalités, ainsi que la modification de l'existant de la solution de facturation et pré comptabilité Evoliz.

B/ Présentation de l'entreprise.

Crée en 2011, cette SAS (Société par Actions Simplifiée) réussit déjà à s'affirmer en tant qu'acteur innovant et dynamique dans le paysage des éditeurs en proposant des solutions facilitant le principe de facturation auprès d'auto-entreprise et désormais de grosses entreprises. Cette entreprise ne comptait, au départ, que seulement 2 employés, leurs fondateurs Olivier et François[1], afin d'arriver à 32 employés avec un recrutement actif de nos jours.

[1] Site d'Evoliz

II/ Résumé de la semaine.

A/ Ressentit et difficulté.

Durant cette 3éme semaine, l'observation s'est amoindri encore et mon implication et de plus en plus demander. En effet, mon implication lors des sessions de ***MOB programming*** est sollicitée et cela est avec plaisir que j'essaie d'apporter ma pierre à l'édifice au sein de l'équipe.

La seule difficulté reste l'anxiété sociale que je me mets tout seul, mais cela reste quelque chose que je dois travailler.

B/ Les activités effectuées.

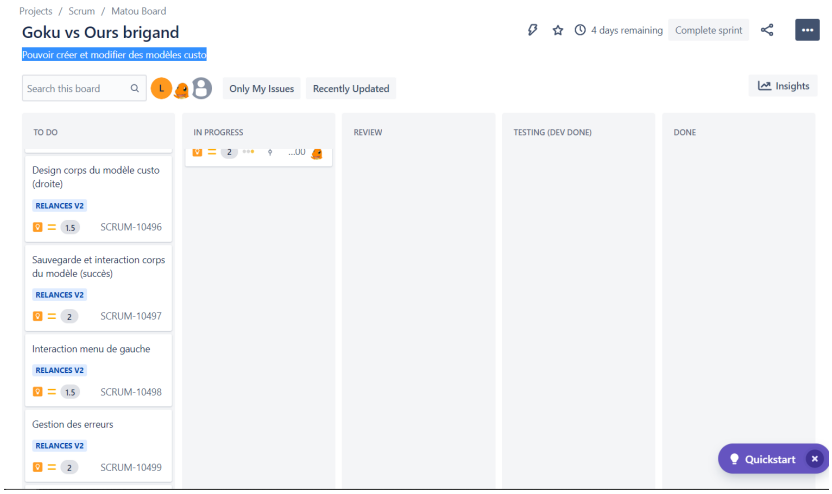
Au niveau des activités, celles-ci restent en lien avec la routine ayant lieu chaque semaine, celle du SPRINT qui reste accès à l'objectif principal qui est *la refonte de l'interface pour créer une relance à partir d'une facture*. Je traiterai cela dans la partie suivante.

Tout comme chaque semaine, à partir de la méthode OKR, un ***SPRINT*** à lieux et se compose en 5 cérémonies : **Planification**, **Daily Scrum Meeting** , **L'affinage, Review et Rétrospective.**

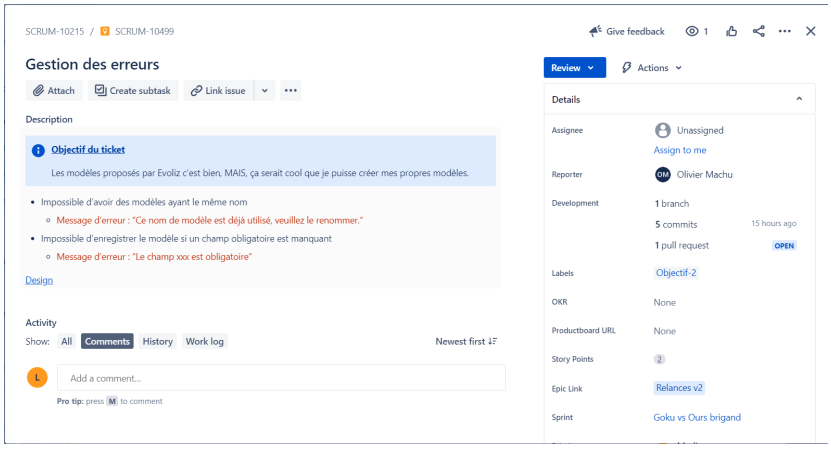
III/ Sprint de la semaine.

A/ L'objectif.

Afin de commencer le sprint de cette semaine, nous allons d'abord entrer dans la phase de planification, dans laquelle nous allons définir des "**user story**" en lien avec l'objectif du sprint qui est : "Pouvoir créer et modifier des modèles de relances customisé".



Afin de réaliser cet objectif, plusieurs "user story" ont été créés afin de permettre la réalisation de celui-ci.



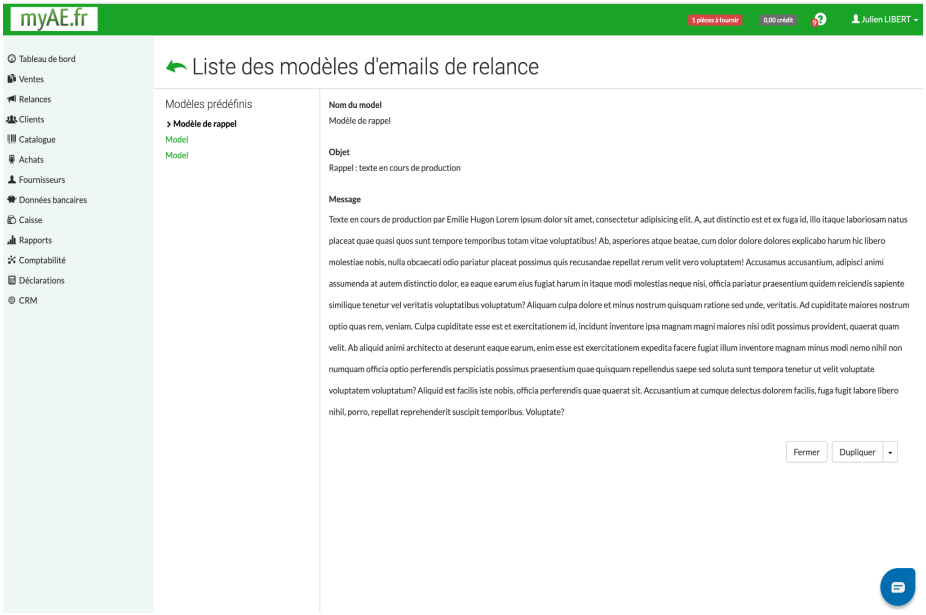
B/ Synthèse des us.

Lors de la planification, les us ont été analysés, potentiellement divisés en plusieurs us et ont reçu une note de difficulté afin de définir la charge de travail, que cela donnerait, comparé à la charge disponible par l'équipe. (Méthode Agile)

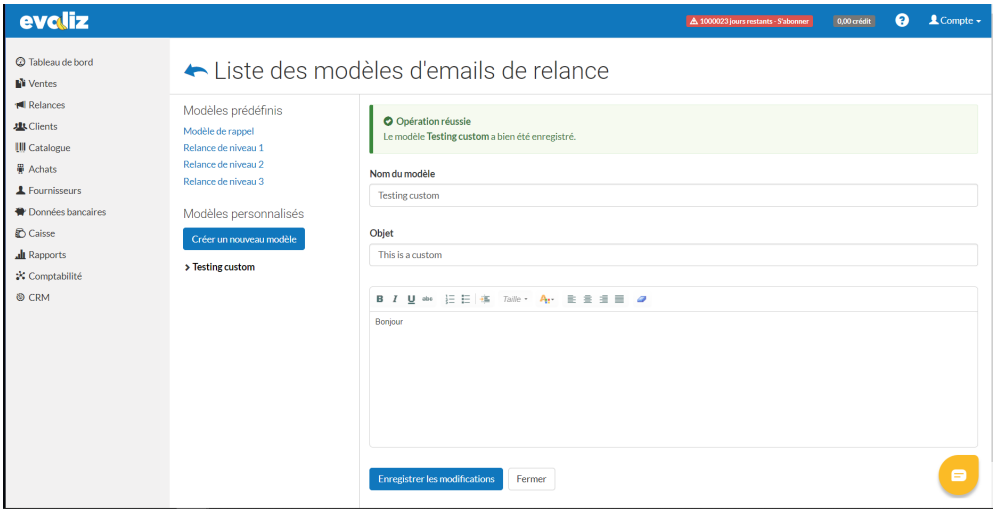
Concernant les us, tous se sont axés à l'interaction de la page de gestion et création de modèles de relances. Le but étant de pouvoir créer, interagir, et modifier des modèles de relances customisés.

Dans les us effectués, ont à tout d'abord commencer par le design de la page (côté gauche puis côté droit, en **mobile last**) s'il y avait des modèles customs pour ensuite ajouter le dynamisme avec php en incluant tous les modèles customs que le client possède ainsi que l'interaction de changement par le biais de **bootstrap 3**. Par la suite, on a ajouté **CKEDITOR** sur l'interface de création de modèle et de modification, étant donné que CKEDITOR été utilisé un peu partout sur notre page, nous l'avons exporté dans une **vue**. Ce qui nous donne le résultat suivant avant - après:

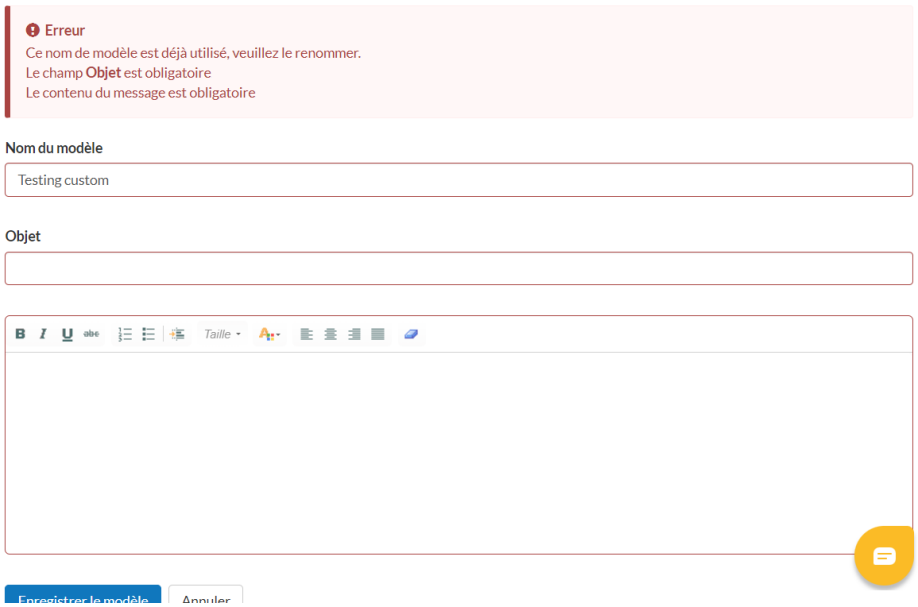
Avant



Aprés



Ensuite, nous nous sommes occupés de l'US concernant la gestion des erreurs (qui fut un peu dure au départ), le but été de faire en sorte qu'en cas d'erreur (si un champ obligatoire manqué, le nom du modèle été déjà utilisé ou une erreur inconnue) la base de données ne soient pas affectés et que l'utilisateur y soit rediriger.




```php
$errors = null;
$templateRecovery = new TemplateRecovery();

if (!empty($_POST)) {
    $postRecoveryTemplateId = filter_input(INPUT_POST, 'recovery-templateid', FILTER_VALIDATE_INT);

    if (empty($_POST['recovery-name'])) {
        $errors['errorFields'][] = 'name';
        $errors['messages'][] = 'Le champ <strong>Nom du modèle</strong> est obligatoire';
    } else {
        $templateNameAlreadyExist = TemplateRecovery::where('LABEL', $_POST['recovery-name'])
            ->where(function ($query) {
                $query->where('COMPANYID', $_SESSION['COMPANYID'])
                    ->orWhereNull('COMPANYID');
            })
            ->where('TEMPLATEID', '<>', $postRecoveryTemplateId)
            ->exists();
        if ($templateNameAlreadyExist === true) {
            $errors['errorFields'][] = 'name';
            $errors['messages'][] = 'Ce nom de modèle est déjà utilisé, veuillez le renommer.';
        }
    }

    if (empty($_POST['recovery-subject'])) {
        $errors['errorFields'][] = 'subject';
        $errors['messages'][] = 'Le champ <strong>Objet</strong> est obligatoire';
    }

    if (empty($_POST['recovery-body'])) {
        $errors['errorFields'][] = 'body';
        $errors['messages'][] = 'Le contenu du message est obligatoire';
    }
```

Pour des raisons de confidentialité, je ne pourrais malheureusement pas montrer certaines partir du code, cependant afin de réaliser cela, nous avons défini un ***tableau à 2 dimension*** vide dans lequel on ajoutera chaque erreur détectée dans le post envoyé ($_POST[2]), par la suite nous avons définie que s'il y avait une erreur, il n'y aurait pas d'enregistrement (en conditionnant `$templateRecovery->save();` le save étant en Eloquent la signification de la commande commit en SQL)

Suite à cela, on a ajouté l'affichage de la page avec les modifications que l'utilisateur avait faites, mais en affichant l'erreur.

[2] $_POST DOC

Pour l'affichage de l'erreur, nous nous sommes servie d'une vue crée par l'équipe dans lequel on lui envoie, en paramètre, notre tableau, transformer en string dans lequel chaque valeur est "séparée" par la balise http
, et sont statut "hidden" en false.

```
    if ($errors !== null) {
        View::renderFile(__DIR__ . '/../../src/Common/Templating/Templates/alert-danger.php', [
            'content' => implode('<br>', $errors['messages']),
            'hidden' => false,
        ]);
    }
    ?>
```

Pour terminer, nous avons ajouté une interaction en javascript dans lequel changer d'interface permet de retirer le message d'erreur.

```
$(document).ready(function () {
    const $recoveryTemplatePage = $('#recovery-template-page');
    const $customTemplatesLink = $recoveryTemplatePage.find('.custom-template-link');
    const $successMessage = $recoveryTemplatePage.find('.alert-success');
    // Chercher l'interface  d'erreur
    const $errorMessage = $recoveryTemplatePage.find('.alert-danger');
    const $tabLink = $recoveryTemplatePage.find('.tab-link');
    const $recoveryTemplateMenu = $recoveryTemplatePage.find('#recovery-template-menu');
    const $newRecoveryButton = $recoveryTemplatePage.find('#new-recovery-button');
```

```
/* Affichage d'une nouvelle interface :
 * ==> suppression de l'erreur
 * ==> Suppression de l'ancienne instance de CKEDITOR
*/
$tabLink.on('show.bs.tab', function () {
    $successMessage.hide();
    $errorMessage.hide();
    $recoveryTemplatePage.find('.form-group.has-error').removeClass('has-error');
    const templateId = $(this).data('templateid');
    for (let [ckeditorInstanceKey, ckeditorInstance] of Object.entries(CKEDITOR.instances)) {
        if (ckeditorInstanceKey !== 'recovery-body-' + templateId &&
            ckeditorInstanceKey !== 'new-recovery-body'
        ) {
            ckeditorInstance.destroy();
        }
    }
});
```

IV/ Vocabulaire.

User story: C'est une situation créée par le Product owner dans le but d'expliquer le besoin que le client pourrait avoir où une amélioration à proposer au client.

Mobile last ou first: Methode de developpement d'une page web en responsive, c'est à dire qu'on commence soit par la version mobile (mobile first) ou la version ordinateur (mobile last)

Bootstrap 3: Framework CSS JS permettant la personnalisation facile de sa page web[3].

CKEDITOR: C'est un WYSIWYG (What you see is What you get !) Éditeur, permettant l'écriture de texte personnalisé dans un textarea.

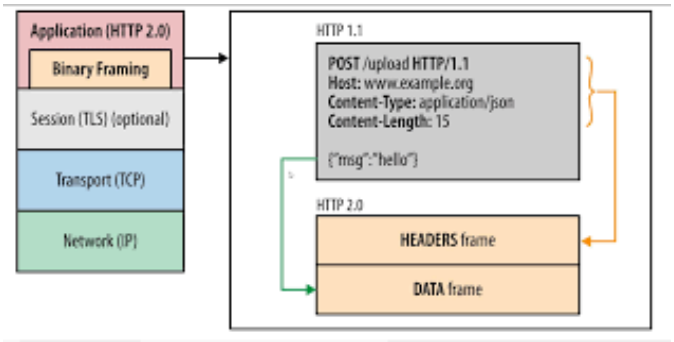
Vue: C'est un fichier dans lequel on y définit un contenu par défaut qui peut être modifié par des paramétre données, ce fichier et par la suite utiliser dans la page afin d'afficher cette vue personnalisé par les paramétre.

Tableau à 2 dimensions: C'est un tableau contenant un 2ème tableau, exemple :

errorsFields		messages	
0	'name'	0	'name missing'
1	'subject'	1	'subject missing'

[3] DOC Bootstrap 3

V/ Approfondissement : Les méthodes de requêtes HTTP.



Il existe plusieurs types de requêtes HTTP, la plus connue et la plus visible est le GET, qu'on peut reconnaître grâce aux “query” situé dans le lien, il se définit par un ? suivis de VARIABLE=VALEUR puis d’un “&” si il y as d’autre valeur. Exemple :

https://monsite.com/findaccount?ACCOUNTID=1234&SUBSCRIBED=TRUE

Il existe donc plusieur requêtes, en voici une liste des plus présent et de leurs descriptives [4] :

GET:

Elle permet de récupérer des information/ressources, elle n’as pas besoin d’un corps mais en réponse elle en contient souvent un.

POST:

Selon le site code-garage : “La méthode POST sert à créer une ressource. Les données pourront être présentes dans les paramètres de l'URL, ou dans le corps de la requête. À noter que le succès d'une requête POST génèrera souvent une réponse 201 (Created), au lieu de la traditionnelle 200 (Success).”

C’est à dire qu’il permet l’envoie d’information au serveurs de façon généralement invisible (non-présente dans le lien HTML, mais présent dans le corps de la requête).

PUT :

Identique à POST à la seule différence qu’elle est idempotente, cela signifie qu’il n’y aura aucun effet de bord en cas d’envoie répétitive de la même requête, ce qui peut malheureusement arriver avec POST.

[4] Liste des méthodes de requêtes HTTP les plus connues.

***DELETE*:**

Cette requête permet de demander la suppression d'une ressource, cela peut être , exemple : La suppression d'une vidéo sur YouTube.

PATCH :

Cette requête applique des modifications partielle au sein d'une ressource (tels que le nom d'une vidéo), elle n'est cependant pas idempotent et peut donc avoir des effets de bords sur l'objet manipulé.[5]

Il est important de connaître l'existence de ses requêtes lorsqu'on travaille avec une API ou une API REST.

Il faut aussi connaître les principaux codes d'erreur que renvoie un serveur web lors d'une requête. En voici une liste sur Wikipédia

[5] La méthode PATCH