BTS SIO
Services Informatiques aux Organisations
LYCÉE BONAPARTE

Rapport d'un BTS SIO

★ *Stage : Evoliz*
Semaine 5

BRUEL LUCAS

09/06/2023

Sommaire

I/ Le stage..... 2
A/ Définition du stage..... 2
B/ Présentation de l'entreprise..... 2
II/ Résumé de la semaine..... 3
A/ Ressentit et difficulté..... 3
B/ Les activités effectuées..... 3
III/ Sprint de la semaine..... 4
A/ L'objectif..... 4
B/ Synthèse des us..... 4
IV/ Vocabulaire..... 6
V/ Approfondissement : Éloquent (Laravel)..... 6
A/ Prérequis :..... 6
B/ Créer une table..... 7
C/ Interaction avec la table..... 8
1. Ajouter des lignes :..... 8
2. Obtenir des information sur la table :..... 9
3. Mise à jours d'une ligne sur une table :..... 9
4. Suppression d'une ligne dans la table..... 10
D/ Les avantages d'Éloquent :..... 10

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] Slte d'Evoliz

II/ Résumé de la semaine.

A/ Ressentit et difficulté.

Durant cette semaine, je vis cette même routine que l'entreprise a mise en place, une routine agréable et qui respecte les employés !

De mon côté, comme chaque semaine, j'ai pu participer au sein de l'équipe de développeurs afin de maintenir et d'améliorer la solution Evoliz.

Coté difficulté, cela à été la compréhension de l'ORM Eloquent du framework php Laravel, cependant, il n'est pas si compliqué que ça et la documentation fournie permet d'en apprendre les bases rapidement.

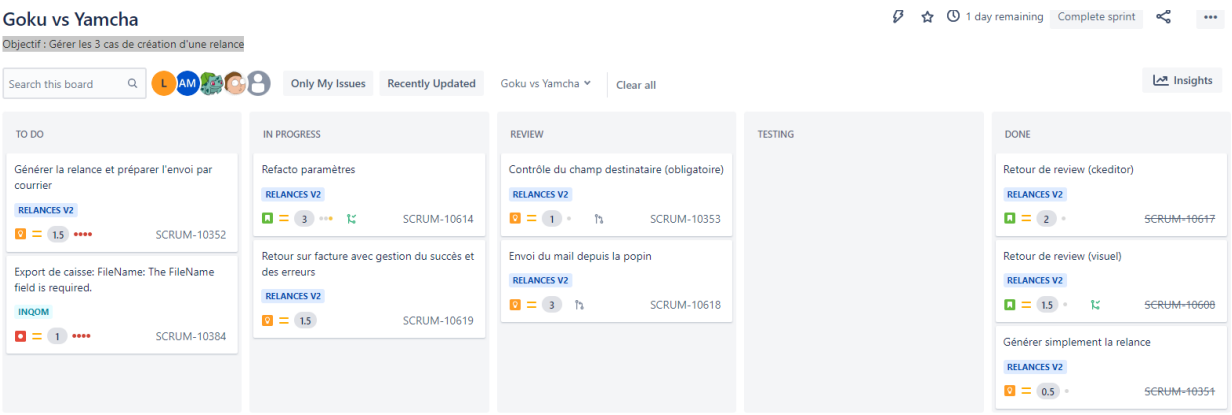
B/ Les activités effectuées.

Au niveau des activités, celles-ci restent en lien avec la routine ayant lieu chaque semaine, celle du SPRINT qui reste axé à l'objectif principal qui est *de gérer les 3 cas de création d'une relance*. 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 : "Gérer les 3 cas de création d'une relance".

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

B/ Synthèse des us.

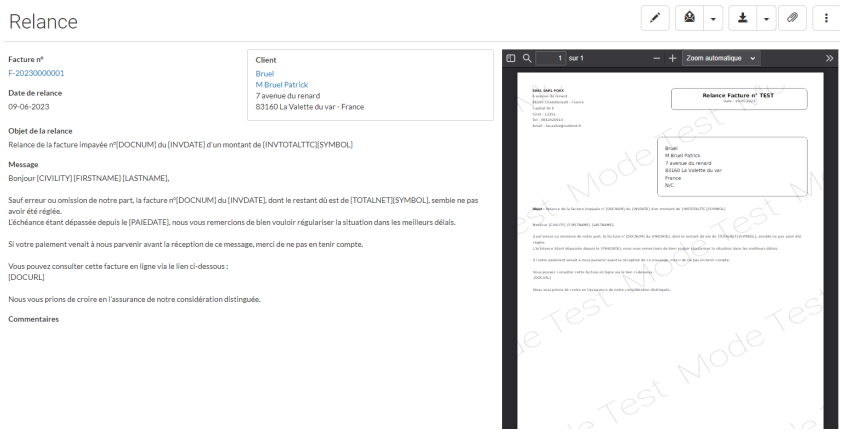
Lors de la planification, les us ont été analysés, potentiellement divisés en plusieurs us et ont reçu une note de difficulté afin de définir la charge de travail, que cela donnerait, comparé à la charge disponible par l'équipe. (Méthode Agile)

Concernant les us, tous se sont axés sur les 3 cas de création d'une relance qui sont présents (Enregistrer, Envoie par mail, Envoie par courrier).

Dans les us effectués, ont à tout d'abord commencer par **refactoriser** le code, afin de l'optimiser et de le rendre indépendant, cela permettra donc la réutilisation de ce code sur d'autres pages.

N'ayant pas fini de tout refactoriser, nous avons décidé d'avancer sur d'autres US du sprint, cela à donc commencé par un garde-fou sur le champ destinataires, afin d'éviter l'interaction inattendue et de protéger ce champ d'interaction.

Ensuite, nous avons géré la création d'une relance par le biais d'une pop-in, celle-ci devait nous rediriger vers la page de la relance enregistrée, nous avons, par la suite, rajouté les actions d'envoyer par mail et par courrier !



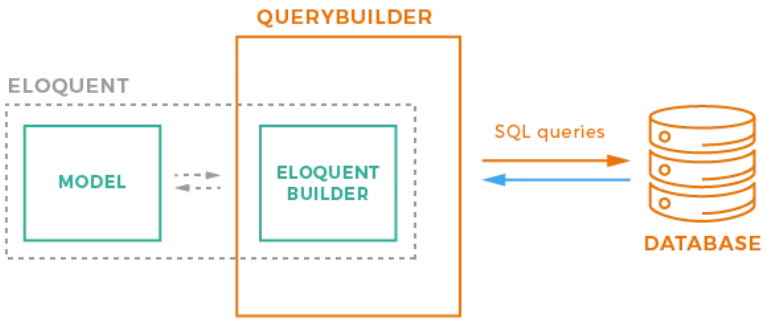
Note : Par soucis de confidentialité, je ne pourrais pas vous montrer de code

IV/ Vocabulaire.

Refactorisation : Amélioration et optimisation du code source d'un projet sans y ajouter de fonctionnalités et corriger des bugs.

V/ Approfondissement : Éloquent (Laravel)

Éloquent est un ORM (Object-Relational mapper) présent dans le framework PHP Laravel, il permet de simplifier l'utilisation d'une BD avec l'utilisation de méthode dans un objet de la classe correspondant à une table dans la DB. Cela permet donc de gagner en efficacité et en efficience.



A/ Prérequis :

Il faudra, avant tout, faire l'installation et la mise en place de Laravel, pour cela, je me sers de la formation gratuite de grafikart.

Configuration d'Éloquent :

Afin de configurer l'ORM, il suffit de modifier le fichier .env, présent à la racine du projet, et modifier les informations de connexion à la Database :

- DB_CONNECTION=mysql
- DB_HOST=127.0.0.1
- DB_PORT=3306
- DB_DATABASE=laravel
- DB_USERNAME=root
- DB_PASSWORD=

B/ Créer une table

Tout d'abord Laravel possède une commande permettant de créer un fichier php dans "./database/migrations/", pour cela, il suffit de faire " *php artisan make:migration NomDeLaTable* ", cela permettra de rajouter des informations dans notre base de données.

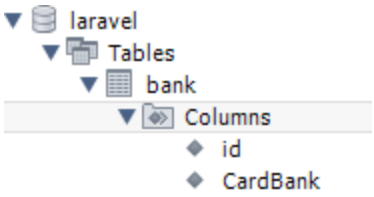
```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('bank', function (Blueprint $table) {
            $table->id(); // crée une colonne Id de type int
            $table->string('CardBank'); // crée une colonne CardBank de type String
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('bank');
    }
};
```

Notre fichier de migration va contenir deux méthodes :

- La méthode "up" qui permet d'expliquer comment générer les tables et les champs demandés.
- La méthode "down" qui permet de vider la DB de la table crée.

On peut donc la customiser et ajouter des gens tels que "id" qui s'auto incrémente, ou une chaîne de caractère unique "CardBank".

Pour ensuite exécuter ce fichier, il suffira d'écrire : " php artisan migrate " .



C/ Interaction avec la table

« C'est bien beau ! On a créé une table, mais en quoi cela va me permettre de faire des requêtes de façon simplifiée ? »

C'est ici que vont intervenir les Modéles, ce sont des classes qu'on pourra instancier afin d'interagir et de modifier la table définie par ce modèle (dans mon exemple la table "bank"). Laravel nous permet de créer ce modèle à partir de la commande "php artisan make:model <nom de la table au singulier>".

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

2 references | 0 implementations
class Bank extends Model
{
    use HasFactory;

    /**
     * @var string
     */
    2 references
    protected $table = 'bank';

    /**
     * @var bool
     */
    0 references
    public $timestamps = false;
}
```

Par défaut:

- La table appelée sera le nom du modèle au pluriel (ajout d'un "s" à la fin), cependant on peut lui définir la table qu'il doit utiliser en ajoutant l'attribut "$table".
- Laravel permet d'indiquer pour chaque ligne sa date de création et de modification, ont peut désactiver cela avec l'attribut "$timestamps"

1. Ajouter des lignes :

Maintenant pour pouvoir interagir avec la table, on va créer un fichier php dans lequel nous allons créer un **objet de la classe Bank** et lui dire que l'attribut , qui doit avoir le même nom que la colonne dans la table, est affecté à la valeur qu'on lui donne.

```php
<?php

use App\Models\Bank;

$bank = new Bank();
$bank->CARDBANK = $_GET['VALUE'];
$bank->save();

?>
```

Il faudra répéter ce processus autant de fois qu'il y a de colonne non-null dans la table. Une fois cela fait, il suffira d'utiliser la méthode save().

	id	CardBank
▶	1	FJDF

2. Obtenir des information sur la table[2] :

Pour les requêtes afin d'obtenir une information, j'appelle mon modèle sans l'instancier et j'appelle sa méthode statique where() dans lequel je lui fournis le nom de la colonne suivis du comparateur (si cela est "=", alors ce paramètre n'est pas requis) et la valeur.

```php
$result = Bank::where('CardBank', 'FJDF')->get();
foreach ($result as $value) {
    echo "$value->id : $value->CardBank <br>";
}
```

Avec where()
1 : FJDF
2 : FJDF
3 : FJDF
4 : FJDF
7 : FJDF
8 : FJDF
9 : FJDF
Avec find() :
5 : SIO

Cela renverra une collection, pour récupérer par exemple l'id de mes ligne, puis je le met dans un foreach et j'appelle la colonne qui va avec.

Il y a la méthode find() qui, en prenant en paramètre la clé primaire, permet de retourner une ligne.

3. Mise à jours d'une ligne sur une table :

Afin de modifier une ligne de ma table, je vais me resservir de ce qu'on à vue dans les 2 parties précédentes.

	id	CardBank
▶	5	SIO
*	NULL	NULL

Dans la database, j'ai inséré à l'id n°5, la valeur SIO à la ligne "CARDBANK".

Afin de modifier cette ligne en "SIO SLAM, je vais récupérer cette ligne avec la méthode find(5), 5 étant la valeur de la clé primaire "id", que je stock d'un une variable, qui sera l'objet de mon modèle, et ensuite, je peut le modifier comme quand on ajoute une ligne dans notre table.

```php
// On obtien la ligne ayant la clé primaire 'id' 5
$ligneSio = Bank::find(5);
// On modifie la valeur de la colonne CardBank
$ligneSio->CardBank = 'SIO SLAM';
// On save notre modification
$ligneSio->save();
```

	id	CardBank
▶	5	SIO SLAM

Note: Il est important d'ajouter le save() à la fin, sinon les modifications ne seront pas prises en compte.

[2] Laravel Eloquent

On peut aussi, selon la documentation d'Éloquent, faire de la mise à jour de masse.

4. Suppression d'une ligne dans la table

Tout comme la mise à jour d'une ligne, on peut la delete de la même façon sauf qu'à la place d'un save(), il faudra utiliser la méthode destroy().

```php
// On obtien la ligne ayant la clé primaire 'id' 5
$ligneSio = Bank::find(5);
// On supprime la ligne
$ligneSio->delete();
```

D/ Les avantages d'Éloquent :

Comme expliqué au début, Éloquent est un ORM, il a la particularité de faciliter l'interaction avec la base de données est donc d'offrir une manipulation des données plus simple, efficace et de masse. De plus, il ajoute une couche de protection contre les injections SQL.