SAÉ S3.A.01 - Développement d’une application

Autour des fractales, galerie d’art

IUT de Lens - Département Informatique

2022-2023

Table des matières

Phase 2 - Gestion d'une galerie d'art

Les attendus

Vous allez travailler en équipe, nous souhaitons vérifier que l'application produite est bien le résultat d'un travail réalisé par chacun des membres de l'équipe. C'est pourquoi dans un premier temps vous allez indiquer dans un fichier `attendus/fonctions.md` de votre projet, la fonction de chaque membre de l'équipe.

L'avancée de votre travail devra être visible dans l'historique des versions de votre projet. Cela signifie que chaque tâche réalisée devra être matérialisée par un ***commit***. Si une tâche est importante, elle pourra être divisée en sous tâche.

D'autre part, nous allons vous donner une liste indicative de tâches ou tickets à réaliser. Chaque tâche ou sous-tâche devra être assignée à un membre de l'équipe et c'est lui qui devra effectuer le commit.

Vous utiliserez pour cela les possibilités offertes par la plateforme *gitlab* de l'université.

Il est préférable d'organiser votre équipe dès le début.

Pensez notamment à définir le rôle de ***validateur*** qui sera la personne qui validera la qualité du code produit par un membre de l'équipe et qui l'ajoutera sur la branche principale (`main`) de votre projet. Ce rôle doit être effectué avec diplomatie en indiquant des critiques constructives (Il n'est jamais agréable, même si le résultat n'est pas correct, d'entendre des critiques négatives).

Dans un fichier `attendus/dialogue.md` vous ajouterez toutes remarques, informations, choix techniques que le correcteur pourra prendre en considération.

Encadrants

— F. Hémery
— T. Hsu
— F. Zimmermann

Évaluation

Pour cette deuxième phase, l'évaluation se fera sur la base des livrables suivants :

— Le code source de votre bibliothèque sur *GitLab*, dont vous déposerez le lien sur *Moodle*. Vous ajouterez pour cela les encadrants de cette SAÉ comme *reporters* sur votre projet.
— Les diagrammes de classes demandés, devront se trouver dans le fichier `attendus/dialogue.md` de votre dépôt *GitLab*, idéalement au format *PlantUML*, ou alors sous la forme d'images `png` dans le répertoire `attendus`.

Présentation de l'application

Dans cette phase préliminaire du projet marathon, nous allons nous immerger dans le monde de l'art en général. Les œuvres pourront être des images numériques comme les images fractales que vous aurez calculées dans la phase précédente.

Nous avons reçu un cahier des charges d'un grand musée national qui souhaite afficher à partir d'un site internet une galerie d'œuvres.

Voici un extrait du cahier des charges que nous a fait parvenir le musée.

Constitution du système d'information

Les entités qui seront utilisées dans notre site sont les suivantes :

— Une **œuvre** est caractérisée par un nom, une description, une date d'inscription au catalogue, un lien vers une présentation multimédia (photo, vidéo, …). Une œuvre est réalisée par un ou plusieurs auteurs.
— Un **auteur** est caractérisé par un nom, un prénom, une nationalité et une date de naissance.
— Les **visiteur**s du site qui se seront authentifiés pourront déposer des commentaires sur le site et construire une liste d'œuvres favorites. Pour s'authentifier, le visiteur devra disposer d'un compte utilisateur (*user*).
— Un **utilisateur** (`User`) est caractérisé par un nom, une adresse mail unique et un mot de passe. Certains utilisateurs, comme les administrateurs du site pourront s'authentifier, mais ne seront pas associés à un visiteur. La distinction entre un utilisateur "visiteur" et un utilisateur administrateur se fera pas un attribut booléen `admin` qui sera faux dans le cas d'un utilisateur "visiteur" et vrai dans le cas d'un utilisateur " administrateur".
— Un **visiteur** est caractérisé par un nom, un prénom et un avatar qui le représente virtuellement.

— Un **commentaire** est donné par un visiteur sur une œuvre, il se compose d'un titre, d'un texte qui constitue le corp du commentaire et peut être accompagné d'une note de zéro à cinq. Une date sera associée à la création du commentaire et une date indiquera la dernière modification du commentaire. Nous allons donner la possibilité au visiteur de faire plusieurs commentaires sur la même œuvre, cela signifie que la Class `Commentaire` ne sera pas une classe association et par conséquent la table `commentaires` aura une clé primaire.
— Le visiteur peut constituer une liste d'œuvres favorites. Ici, contrairement à la relation précédente, une œuvre n'apparait qu'une seule fois dans la liste des favorites, on a donc affaire à une classe association.

1. Proposez un modèle conceptuel de données (*MCD*) à partir des caractéristiques indiquées dans l'extrait du cahier des charges précédent (fichier `diag_mcd.plantuml` ou une image `diag_mcd.png`). Utilisez un diagramme de classes UML pour représenter le MCD.

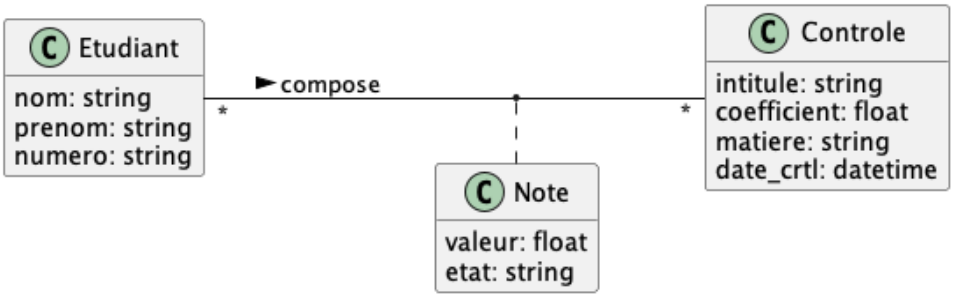

FIGURE 1 – Exemple modèle MCD

2. Proposez un modèle logique de données (*MLD*) qui permet d'adapter le MCD vers une base de données relationnelle. Utilisez à nouveau un diagramme de classes UML pour représenter le MLD qui contiendra cette fois toutes les tables relationnelles ainsi que les clés primaires (fichier `diag_mld.plantuml` ou une image `diag_mld.png`).

   On indiquera sous forme de note, les contraintes référentielles ou autres qui existent entre les tables et contraintes de domaine pour les attributs.

Les deux fichiers précédents seront ajoutés au répertoire `attendus`.

L'application web d'un musée virtuel

Nous allons poser les bases d'un POC (*Proof Of Concept*) de notre musée virtuel.

Pour cela nous allons utiliser le cadriciel Laravel qui a été étudié en cours.

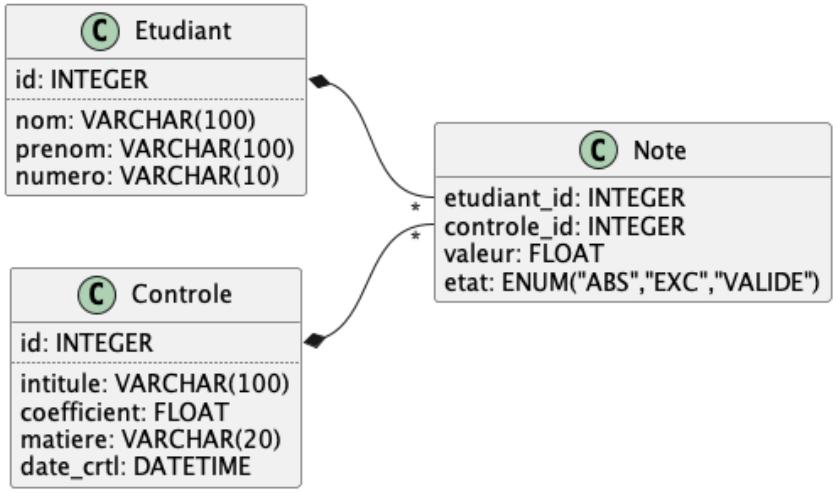

FIGURE 2 – Exemple modèle MLD

Dans notre projet, nous aurons besoin d'utiliser un système de gestion de bases de données relationnelles (*SGBDR*), dans cette phase préliminaire, nous allons privilégier la simplicité, on utilisera le SGBDR SQLite.

Nous aurons besoin d'installer un mécanisme d'identification et d'authentification. A nouveau, dans le cadre de ce projet préliminaire, nous allons limiter ce mécanisme à

- la possibilité pour un visiteur de créer un compte utilisateur
- la possibilité pour un visiteur de s'authentifier
- les administrateurs auront des droits particuliers

Cela signifie, que l'on ne gérera pas la possibilité pour un utilisateur de réinitialiser son mot de passe.

Nous allons utiliser la librairie Fortify pour ajouter une couche authentification à notre musée virtuel. Fortify permet de gérer les mécanismes d'identification et d'authentification sans proposer les vues (`login` et `register`) de dialogue avec l'utilisateur. Son avantage est d'être simple à mettre en œuvre, son inconvénient, c'est qu'il faudra construire les vues.

Après avoir géré le système d'information de notre projet et éventuellement utiliser une librairie bootstrap pour gérer le style des pages de notre application, nous allons suivre la documentation donnée par Laravel pour utiliser Factory.

Les étapes de construction de notre projet

Étape 1 : Le système d'information

Nous allons commencer par créer un projet à partir du cadriciel Laravel.

1. Créez un projet Laravel `poc-sae3-01-grpXX` avec `XX` votre numéro de groupe pour la SAÉ3.A.01.
2. Créez un fichier `.env` et configurez la connexion de votre projet avec un *SGBDR SQLite*.
3. Créez les classes modèles que vous allez utiliser et ajoutez dans ces classes, les fonctions qui matérialisent les relations entre classes.
4. Créez les fichiers de migration pour créer les tables dans votre base de données (Si vous n'êtes pas sûr de votre système d'information, il peut être judicieux de demander un avis à un enseignant).
5. Créez les fichiers de fabrique de données aléatoires pour stocker des enregistrements dans les différentes tables.
6. Créez les classes `Seeder` qui utilisent les fabriques de données.
7. Tester le bon fonctionnement de la mise en place de votre système d'information et vérifiez qu'il y a des données dans toutes les tables.

Étape 2 (optionnelle) : installation de la librairie bootstrap

Cette étape est optionnelle si vous souhaitez gérer le style par vous-même, vous pouvez passer à l'étape suivante directement.

Lors du projet marathon, le style de votre application web sera géré en partenariat avec des étudiants en BUT MMI. Certaines ressources, dans leur formation, introduisent des notions de design et de conception de charte graphique qui ne sont pas abordées en BUT Informatique. Ici l'utilisation d'une librairie externe vous permettra de gagner un peu de temps.

Récupération des dépendances pour la gestion du front avec bootstrap

Bootstrap vous permet de gérer vos styles plus simplement et propose de nombreux widgets qui nous permettent d'aller plus vite dans la création des pages de notre site.

```
npm install bootstrap
npm install sass
npm install path
```

Les fichiers de configuration

1. Création d'un répertoire `resources/scss`
2. Création d'un fichier `resources/scss/_variables.scss`

   Contenant par exemple les variables suivantes :

```
$body-bg: #f8fafc;

$primary: #26A69A;
```

3. Création d'un fichier `resources/scss/app.scss` : Après avoir créé le répertoire , éditez le fichier `resources/scss/app.scss` avec le contenu suivant :

```
@import url('https://fonts.googleapis.com/css?family=Nunito');
@import 'variables';
@import '~bootstrap/scss/bootstrap';
```

4. Modification du fichier `resources/js/app.js` avec le contenu suivant :

```
import './bootstrap';

import * as bootstrap from 'bootstrap'
```

5. Modification du fichier `vite.config.js` avec le contenu suivant :

```

import {defineConfig} from 'vite';
import laravel from 'laravel-vite-plugin';
import path from 'path'

export default defineConfig({
  plugins: [
    laravel({
      input: ['resources/scss/app.scss',  'resources/css/app.css',
           'resources/js/app.js'],
      refresh: true,
    }),
  ],
  resolve: {
    alias: {
      '~bootstrap': path.resolve(__dirname, 'node_modules/
         bootstrap'),
    }
  },
});
```

6. Modification du modèle de vue blade pour utiliser la configuration Vite

   Par exemple une page accueil.blade.php :

```blade
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ config('app.name', 'Laravel') }}</title>

    <!-- Fonts -->
    <link rel="stylesheet" href="https://fonts.bunny.net/css2?family=Nunito:wght@400;600;700&display=swap">

    <!-- Scripts -->
    @vite(['resources/scss/app.scss','resources/css/app.css','resources/js/app.js'])
</head>
<body>
<div class="container py-4 px-3 mx-auto">
    <h1>Hello, Bootstrap and Vite!</h1>
    <button class="btn btn-primary">Primary button</button>
</div>
</body>
</html>
```

> **Note :** Ne pas oublier de mettre à jour le *front* après ces modifications en utilisant la commande `npm run build`.

Étape 3 Installation de Fortify

Nous allons maintenant ajouter la possibilité aux visiteurs de s'identifier. Pour cela ils auront la possibilité de se créer un compte, c'est-à-dire un enregistrement dans la table `users` associé avec un enregistrement dans la table `visiteurs`.

Dans un premier temps nous allons installer la librairie Fortify. Pour cela vous pouvez suivre les explications qui vous seront données dans la suite ou vous aider de la documentation disponible sur le site Laravel.

— Configuration de fortify

```
composer require laravel/fortify
php artisan vendor:publish --provider="Laravel\Fortify\FortifyServiceProvider"
```

— Enregistrer Fortify comme un service de gestion de l'authentification :

Ajoutez la ligne suivante dans le fichier `config.app.php`

```
'providers' => [
    /* ... */
    /*
     * Package Service Providers...
     */

    App\Providers\FortifyServiceProvider::class,

    /*
     * Application Service Providers...
     */
    /* ... */
]
```

— Déclaration des possibilités de Fortify dans le fichier `config/fortify.php`

```
    'features' => [
        Features::registration(),
        Features::resetPasswords(),
        // Features::emailVerification(),
        //         Features::updateProfileInformation(),
        //         Features::updatePasswords(),
        //         Features::twoFactorAuthentication([
        //             'confirm' => true,
        //             'confirmPassword' => true,
        //             // 'window' => 0,
        //         ]),
    ],
```

— Création des vues `accueil`, `home`, `login` et `register`
On utilisera le moteur de vue `blade` pour factoriser le modèle des vues de notre POC.

— La vue **accueil** (`accueil.blade.php`) sera affichée à l'aide de l'url http://localhost:8000/. La vue contiendra un lien vers la vue de connexion `login` et vers la vue de création d'un nouvel utilisateur `register`.

— La vue **home** (`home.blade.php`) sera affichée à l'aide de l'url http://localhost:8000/home. Cette vue ne sera accessible qu'aux visiteurs identifiés, elle contiendra un lien vers la fonction de déconnexion `logout`.

— La vue **login** (`auth/login.blade.php`) qui contiendra un formulaire de connexion (champs `email` et `password`). En cas de validation de la connexion, le visiteur sera redirigé vers la vue **home**.

— La vue **register** (`auth/register.blade.php`) qui contiendra un formulaire de création d'un utilisateur (`User`) et du visiteur associé. Le formulaire contiendra les champs nom, prénom, adresse mail, mot de passe, nationalité et date de naissance.
En cas de validation de la saisie du formulaire, l'application devra créer un nouvel utilisa-

teur dans la table `users` et un nouveau visiteur dans la table `visiteurs` associé avec l'utilisateur. L'avatar du visiteur sera initialisé par un lien vers une image par défaut.

— Utilisation des vues précédentes par Fortify

Ajoutez les instructions suivantes dans la fonction `boot` de la classe `FortifyServiceProvider` dans le fichier `app/Providers/FortifyServiceProvider.php`

```
1   Fortify::loginView(function () {
2       return view('auth.login');
3   });
4
5   Fortify::registerView(function() {
6       return view("auth.register");
7   });
```

— Le formulaire dans la vue `login` servira à l'authentification d'un visiteur. En cas de succès, le visiteur sera redirigé vers la vue `home`.

— Le formulaire dans la vue `register` sera traité par la méthode `create` de la classe `CreateNewUser` dans le fichier `app/Actions/Fortify/CreateNewUser.php`. Il sera nécessaire de modifier la méthode `create` de façon à permettre la création d'un visiteur associé à un utilisateur.

Il n'est pas nécessaire pour le développeur de créer les routes vers les vues `login` et `register`, ces routes ont été ajoutées par Fortify. Il faut uniquement ajouter la route vers la vue `home` dont l'accès devra être réservé aux visiteurs connectés.

> Note : la commande `php artisan route:list` affiche les routes connues pour votre application.

— Le menu de départ

Comme indiqué plus haut, la vue doit permettre à un visiteur pas encore connecté de se connecter ou de créer un nouveau compte et permettre à un visiteur connecté de se déconnecter. Le code qui suit donne un exemple des instructions qui permettront de gérer cette possibilité :

```
1  <nav>
2   <ul>
3       @guest
4           <li><a href="{{ route('login') }}">Login</a></li>
5           <li><a href="{{ route('register') }}">Register</a></li>
6       @else
7           <li> Bonjour {{ Auth::user()->name }}</li>
8           @if (Auth::user())
9               <li><a href="#">Des liens spécifiques pour
                    utilisateurs connectés..</a></li>
10          @endif
11          <li><a href="{{ route('logout') }}"
12                 onclick="event.preventDefault(); document.
                       getElementById('logout-form').submit();">
13                  Logout
```

```
14                </a></li>
15            <form id="logout-form" action="{{ route('logout') }}"
                  method="POST" style="display: none;">
16                {{ csrf_field() }}
17            </form>
18        @endguest
19    </ul>
20 </nav>
```

1. Ajoutez la dépendance Fortify à votre projet.
2. Configurez Fortify comme outil de gestion de l'identification des utilisateurs de l'application.
3. Ajoutez les vues `accueil`, `home`, `login` et `register` à votre application. Les vues utiliseront le moteur de vues Blade.
4. Gérez l'identification et la création d'un nouvel utilisateur associé à un visiteur.
5. Testez votre application et ses nouvelles possibilités.

Gestion des fichiers stockés sur le serveur

Avant de créer les pages de notre application, nous allons préparer l'application pour stocker des fichiers (avatars ou œuvres) du côté du serveur, pour cela nous allons reprendre le TP laravel qui donne quelques explications sur cette mise en place.

Cette préparation consiste, entre autres choses, à

— Modifier le répertoire de stockage des fichiers téléversés (*uploader*) à l'aide de la variable `FILESYSTEM_DISK`,
— Ajouter un lien entre le répertoire de `storage` et le répertoire **`public`**.

Quand ces modifications auront été effectuées, il sera possible de téléverser des fichiers et de les afficher dans une page.

Mise en place des pages de notre application

La liste des œuvres

Dans la page d'accueil, nous allons afficher les œuvres du musée dans notre galerie d'art. Cet affichage est ouvert au visiteur anonyme. Il faut imaginer un mécanisme qui vous permet de choisir entre les différentes possibilités proposées ci-dessous.

1. Affichez la liste des œuvres (nom, date d'ajout)
2. Filtrez les oeuvres par auteur

3. Affichez les 5 œuvres les plus récentes
4. Affichez les 5 œuvres les mieux notées (notes données dans les différents commentaires postés sur l'œuvre)

Les détails d'une œuvre

A partir de la page précédente, nous souhaitons pouvoir afficher les détails d'une œuvre.

Cela signifie pour l'utilisateur anonyme :

1. Affichez les informations d'une œuvre (nom, description, date d'ajout)
2. Affichez, le, les auteurs
3. Affichez l'image associée au lien média
4. Affichez les commentaires triés par ordre chronologique (du plus récent au plus ancien)

Dans le cas où l'utilisateur est connecté :

1. Affichez un élément qui indique si cette œuvre fait partie de la liste de ses favorites
2. Ajouter ou supprimer cette œuvre dans la liste des favorites

Affichage et modification des informations d'un visiteur

Cette page est réservée à un utilisateur connecté

1. Affichez les informations du visiteur
2. Affichez la liste des œuvres favorites
3. Affichez la liste des commentaires écrits par le visiteur
4. Modifiez l'avatar d'un visiteur (par téléversement (`uploading`) d'une image sur le serveur)

La gestion des commentaires d'une œuvre

Dans la page qui affiche les détails d'une œuvre, on ajoutera la possibilité à un utilisateur connecté de

1. Créer un commentaire
2. Modifier un commentaire (créé par le visiteur connecté ou l'administrateur)
3. Supprimer un commentaire (créé par le visiteur connecté ou l'administrateur)

La création d'un composant

Dans la page qui affiche les détails d'une œuvre, on ajoutera un composant `statistiques` qui contient les informations suivantes :

1. La note moyenne attribuée à l'œuvre
2. La note la plus haute, la note la plus basse
3. Le nombre de notes attribuées à l'œuvre
4. Le nombre de visiteurs qui ont ajouté l'œuvre dans ses favorites