Aix-Marseille Université

M3103 - Programmation Web côté serveur

Réalisation d'un site de partage et de création de dessins

Auteurs :
Lucien Aubert
Tom Henoch

Enseignants :
Brett Desbenoit
Laurent Carmignac

Table des matières

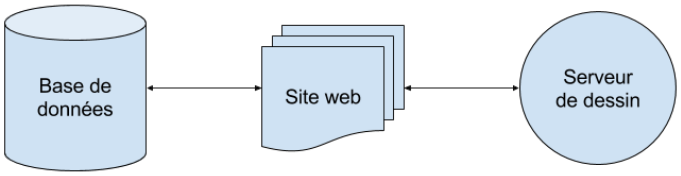
1 Introduction

L'utilisateur sera en mesure de dessiner avec des outils basiques (crayon, carré, cercle, lettrages) et de partager sa création. Les dessins ainsi créés sont modifiables par plusieurs utilisateurs.

Les utilisateurs pourront commenter les dessins et les marquer comme "J'aime".

2 Structure

L'application est composée de trois modules qui interagissent :
- un serveur web (site + base de données)
- un serveur spécifique à l'édition de dessin
- une application client servie par le serveur web



La base de données n'est accessible que par le site web, doté d'une API type REST permettant au serveur de dessin de lui passer des requêtes.

2.1 Serveur Web

Le site web est développé avec le framework [1]Symfony 3 et utilise une base de données MariaDB. Il est hébergé sur un serveur Apache 2.4.

Le moteur de template utilisé est Twig et l'ORM utilisé est Doctrine2.

Ce serveur joue les rôles d'interface utilisateur et d'interface entre le serveur de dessin et la base de données.

C'est lui qui donne la permission au serveur de dessin de démarrer une connexion avec un client.

2.1.1 Connexion à la base de données

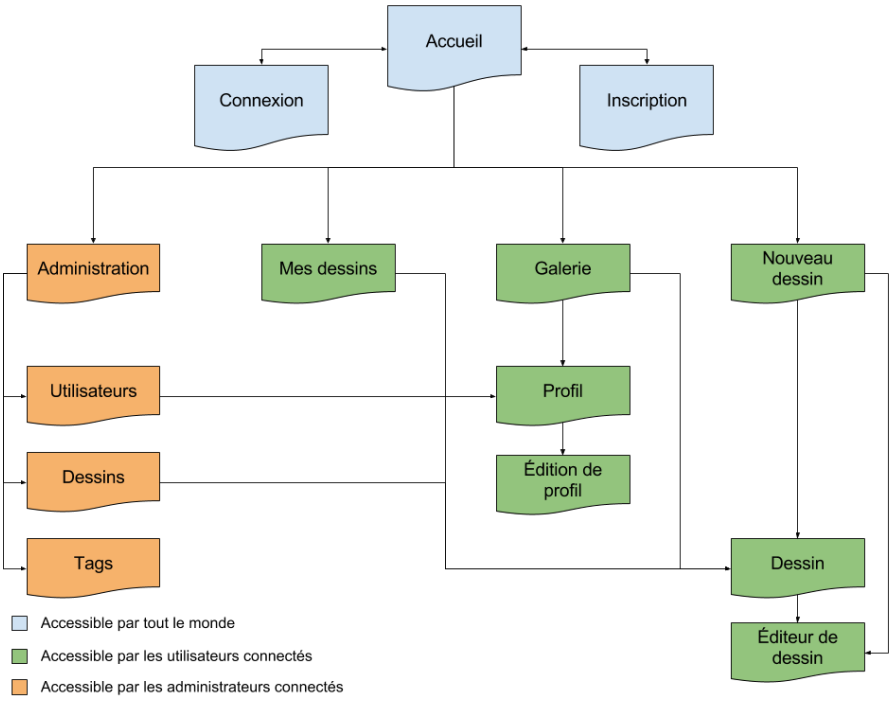
Les paramètres de connexion à la base de données se trouvent dans le fichier `app/config/parameters.yml`.

Afin d'exécuter des requêtes SQL sur le serveur de base de donnée MariaDB nous utilisons la bibliothèque Doctrine fournie avec notre framework.

2.1.2 Fonctions

Pour l'utilisateur, le site permet de
- s'inscrire
- se connecter (authentification par nom et mot de passe)
- créer un nouveau dessin
- consulter la galerie
- rechercher un dessin par nom
- rechercher un dessin par tag
- modifier son profil
- modifier ses dessins (titre, auteurs, tags)

2.1.3 Plan du site



Les pages protégées redirigent vers la page de connexion si l'utilisateur n'est pas connecté et vers la page d'accueil quand il tente d'accéder à du contenu qui ne lui est pas accessible.

Remarque Symfony aidant, il nous est possible de vérifier si l'utilisateur nous a donné le cookie d'une session grâce à cette simple condition, `$this` étant notre controleur.

```
1 if(!$this->getUser())
2   return $this->redirectToRoute('login');
```

2.1.4 Espace d'administration

Le premier utilisateur (avec l'id 1), ou super-administrateur, et tous les administrateurs qu'il nommera peuvent accéder à un espace d'administration à partir duquel il peuvent supprimer et modifier les autres utilisateurs (et les autres administrateurs dans le cas du super-administrateur), les dessins et les tags.

L'accès à cet espace est restreint grâce au membre booléen `is_admin` du modèle `User`. Si il est *vrai* la requête est acceptée, sinon le client est redirigé vers la page d'accueil.

```
1  if(!$user->getIsAdmin())
2    return $this->redirectToRoute('homepage');
```

2.1.5 Améliorations

Intégration d'un éditeur Javascript : L'application de dessin a été développée et intégrée au site. Voir le chapitre concerné plus bas.

Catégories avancées : Pour chaque dessin les auteurs peuvent ajouter/supprimer des *tags*. Il est possible d'afficher les dessins affiliés à un tag en cliquant sur l'un d'entre eux.

Notation : Un système de *J'aime* a été implanté. Chaque utilisateur peut aimer ou non un dessin. Les dessins sont classés par nombre de *J'aime* dans la galerie.

Internationalisation : Le site est traduit grâce au système de localisation Symfony.

Statut des utilisateurs : Tous les noms d'utilisateurs écrits sur les pages du site sont accompagnées d'un macaron de couleur indiquant si l'utilisateur est connecté (vert) ou non (gris). Pour se faire un DATETIME est mis à jour en base de données à chaque action effectuée par l'utilisateur. Si cette valeur est plus éloignée que 2 minutes avant maintenant, l'utilisateur est connecté.

2.1.6 Routage

La chaîne `'login'` correspond ici à la route vers laquelle rediriger le client. Le système de routing du framework permet de définir des routes en leur attribuant un nom, des arguments et une méthode qui sera exécutée quand la route sera demandée par un client. Ces définitions sont faites sous forme de commentaires formatés comme suit.

```
1    /**
2    * @Route(
3    *   "/sketch/{sketchId}",
4    *   requirements={"sketchId": "\d+"},
```

```
5    *   name="sketch"
6    * )
7    */
```

Les fichiers source sont ensuite traités par Symfony et les commentaires parsés pour produire un module de routage sur mesure.

2.1.7 Feuilles de style et UX

Afin de garantir un résultat similaire sur un maximum de navigateurs nous avons choisi d'utiliser un bootstrap appelé Kickstart. Cet outil nous apporte un CSS basique et un module Javascript permettant notamment l'affichage de notifications colorées dans le coin supérieur droit de l'écran.

2.1.8 Ajax

Nous avons privilégié des interfaces avancées pour certains formulaires (champs d'édition de tags, d'auteurs, de titre). Pour se faire nous avons eu recours à l'utilisation de `XMLHttpRequest`, une object Javascript qui permet de faire envoyer des requêtes HTTP par le client. Le client envoie donc des requêtes `GET` vers le serveur pour signaler l'ajout ou la suppression d'un tag ou d'un auteur, le serveur envoie sa réponse sous forme d'un fichier JSON avec une message d'erreur ou de succès qui sera parsé par le client pour afficher une notification.

Deux modules ont été développés pour permettre la validation des ces champs via des requêtes Ajax. Il s'agit des fichiers `web/js/tagInput.js` et `web/js/fastInput.js` qui sont initialisés dans les vues qui les utilisents (`app/Resources/views/show`$_s$*ketch.html.twigpa*

2.2 Serveur de dessin

Cet élément est développé en Javascript et est supposé être lancé avec le moteur [2]NodeJS. Ce serveur fait la liaison entre l'application client de dessin et le serveur web. Elle s'occupe de transmettre les modifications aux utilisateurs qui modifient un même dessin.

L'utilisation du module Javascript Socket.io nous a permis de faire la liaison entre le client et le serveur de dessin.

2.2.1 Authentification

La consultation et la modification de la base de données par l'application client est permise par l'authentification de ce dernier auprès du serveur de dessin.

Le client se connecte au site web pour accéder à la page d'édition de dessin. La page lui est envoyée avec un jeton propre à cet utilisateur et à ce dessin. Le client lance l'application Javascript qui envoie le jeton au serveur de dessin. Le serveur de dessin s'occupe de vérifier le jeton en demandant sa validation au serveur web. Si

le jeton est bon le client peut modifier le dessin et récupérer les changements faits dessus par les autres clients.

L'API que sert le serveur web n'acceptera aucune requête si le jeton envoyé avec est invalide. Ainsi, chaque commande passée par l'application client est authentifiée et, pourtant, passe par le serveur de dessin, qui peut donc assumer son rôle de diffuseur (*broadcast*).

2.2.2 Communication

Pour communiquer tant avec le serveur web qu'avec le client nous utilisons le format JSON, dont l'utilisation est possible nativement en Javascript. Les retours du serveur web, les requêtes en provenance de l'application client ainsi que les retours et requêtes du serveur de dessins sont passées dans ce format, par le protocole HTTP.

2.2.3 Upload de fichiers malveillants

L'upload de fichier étant un point sensible d'une application web nous avons décidé d'endiguer le risque de présence de fichier malveillants sur le serveur en implantant dans le serveur de dessin les mêmes classes que dans l'application client afin de redessiner sur le serveur l'image produite par les utilisateurs. Ainsi aucun fichier de l'utilisateur ne peut être téléchargé sur le serveur.

2.3 Application client

L'application client consiste en une page HTML dans laquelle sont chargés différents modules Javascript regroupés dans web/js/Sketcher grâce au fichier web/js/Sketcher/Sketcher.js.

Elle se charge de permettre l'édition d'un dessin à partir des données (jeton de connexion, dimensions et données du dessin) présente dans la page. Elle communique directement avec le serveur de dessin à l'aide du module Javascript Socket.io après lui avoir envoyé le jeton de connexion.

2.3.1 Canvas

L'application client est basée sur la technologie canvas dont les navigateurs récent implémentent l'API de dessin. Nous dessinons dans un nœud DOM <canvas> à l'aide de méthodes telles que

```
1 ctx.fillRect(0, 0, 20, 20);
2 ctx.strokeRect(0, 0, 20, 20);
```

2.3.2 Sélection des couleurs

Nous avons jugé préférable d'utiliser un module externe, [3]ColorPicker de Peter Dematté, pour permettre la sélection des couleurs.

2.3.3 Format de données

Afin de transmettre les nouveaux objets du dessin et de recevoir ceux diffusés par le serveur de dessin, nous utilisons un objet JSON.

```
{
  width: 200,  // Pixels
  height: 200,
  layers: [
    {
      name: "Background",
      objects: [
        {
          "type": "Circle",
          "data": {
            "options": {
              "thickness": 5,  // Pixels
              "fill": true,
              "stroke": true
              },
            "p1": { "x": 100, "y": 100 },
            "p2": { "x": 150, "y": 150 },
            "stroke_color": "rgba(0, 0, 0, 1)",
            "fill_color": "rgba(95, 195, 29, 1)"
          }
        }
      ]
    }
  ]
}
```

Voici par exemple l'objet transmis par le client au serveur de dessin lors de l'édition d'un dessin de 200 pixels de large sur 200 pixels de haut contenant un cercle vert bordé de noir.

3 Difficultés rencontrées

La prise en main d'un framework aussi fourni que Symfony a été fastidieuse bien que les documentations du framework, de Twig et de Doctrine soient abondantes.

Le développement d'application avec communication asynchrone comme c'est le cas du serveur de dessin et de l'application client était un exercice nouveau. Il nous a fallu trouver des solutions (bibliothèques, serveurs) nous permettant d'atteindre nos objectifs et en apprendre l'utilisation.

4 Conclusion

La création de cette application nous a permis de découvrir le framework Symfony 3 ainsi que la technologie [4]WebSocket. L'utilisation du Javascript pour le développement de l'outil de dessin nous a appris beaucoup sur ce langage, tant côté serveur que côté client.

Le déploiement de l'application a été un défi que nous avons relevé en nous documentant sur le serveur web que nous utilisions, Apache 2.4, ce qui nous a permis d'approfondir les connaissances que nous avons acquis en cours de réseau.

Références

[1] Wikipédia. Symfony.
https://fr.wikipedia.org/wiki/Symfony.

[2] Wikipédia. Nodejs.
https://fr.wikipedia.org/wiki/Node.js.

[3] Peter Dematté. Colorpicker v1.0.
http://www.dematte.at/colorPicker/.

[4] Mozilla Developper Network. Websockets api.
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.