Outil d'aide à la classification automatique de tessons

Herrmann Emmanuel

4 Avril 2016 – 30 Juin 2016

Stage Licence 3 – Ingénierie Informatique

Maître de stage : Matthieu Exbrayat

Table des matières

REMERCIEMENTS

Plusieurs personnes, ainsi que des sites internet, m'ont aidé à la réalisation de l'application. Je voudrais remercier en premier lieu Matthieu Exbrayat, maître de stage, pour son implication dans la conception de l'application, tant au niveau de la conception de l'architecture que de problèmes plus précis, ainsi que des nombreuses heures passées dans son bureau à comprendre certains aspects de Symfony ou des requêtes JQuery. Sebastien Jesset, archéologue, a aussi été prédominant dans la réalisation de mon stage, apportant son expérience archéologique et définissant les besoins auxquels l'application devait répondre. Les forums de developpez.net et surtout stackoverflow m'ont aussi aidé à résoudre nombre de problèmes autour de Symfony et Twig, ainsi que dans une moindre mesure OpenClassroom pour les bases.

INTRODUCTION

Mon stage s'est déroulé dans le cadre du projet ARCADIA. Il s'agit d'un projet financé par la région Centre et regroupant trois organismes : le pôle d'archéologie d'Orléans, à l'origine du besoin, le laboratoire PRISME (Laboratoire Pluridisciplinaire de Recherche Ingénierie des Systèmes, Mécanique, Énergétique) de Polytech pour son savoir faire dans le traitement du signal et de l'image, et le LIFO (Laboratoire d'Informatique Fondamentale d'Orléans) pour ses compétences dans les techniques avancées de classification. Le but de ce partenariat entre ces trois acteurs vient d'un besoin archéologique. En effet, différentes campagnes de fouilles ont mis en évidence la présence d'un centre important de production de poteries dans la ville de Saran. Des tessons de poteries ont été retrouvés en grand nombre, et la plupart sont ornés de motifs. Ces motifs ont été tracés avec des instruments (molettes) qui imprimaient dans l'objet une série de motifs, que l'on répétait en faisant tourner cette molette sur différents endroits de la poterie. On peut identifier le potier qui a fait ces objets car il utilisait souvent le même motif de molette, et donc dater plus ou moins précisément le tesson.

Malheureusement, les archéologues se sont heurtés à plusieurs problèmes : d'une part, une partie des tessons retrouvés sont en mauvais état, et la lecture des motifs peut se révéler difficile ; d'autre part, les molettes étant faites en bois, elles s'abîmaient assez vite, rendant le dessins moins lisible ou sensiblement différent du même dessin mais en début de vie de la molette ; et enfin, le dernier problème vient du potier lui-même, qui peut avoir gravé ses poteries avec des imperfections (glissement, défaut dans l'argile, …). Le travail des archéologues est aussi fastidieux pour une autre raison : le traitement des tessons de poterie. En effet le travail de l'archéologue se divise en trois parties : Mouler le tesson dans de la pâte à modeler, encrer cette pâte pour obtenir un négatif du décor, puis enfin numériser et vectoriser ce motif pour obtenir une version numérique classifiable. Cette méthode étant assez longue, elle a vite trouvé ses limites devant le nombre important de tessons à classifier (plusieurs dizaines de milliers). C'est ainsi que le projet ARCADIA a été créé, afin de développer une méthode d'extraction et de classification automatique de ces tessons, afin de pouvoir soulager le travail des archéologues et de pouvoir classifier plus vite cette partie de notre patrimoine.

Le sujet de mon stage consiste à fournir un outil permettant une gestion centralisée des tessons et de leur classification (que ce soit pour l'enregistrement des images ainsi que de leurs informations, la saisie et la validation des résultats des classifications, …). Suivant les besoins formulés, cet outil a pris la forme d'une application web développée sous Symfony 3, un framework PHP suivant la structure MVC. Le moteur de templates Twig a été utilisé pour générer les pages HTML, Doctrine pour gérer la base de données et Bootstrap pour l'habillage CSS de l'application. Actuellement, cette application permet d'enregistrer des tessons dans la base de donnée avec leurs informations et numérisations correspondantes, de parcourir l'ensemble de tessons avec différents critères de recherche, de modifier les informations d'un tesson ou d'en télécharger des

numérisations, ainsi que de les classifier à la main. Il sera peut-être possible plus tard de les classifier automatiquement via un algorithme, une partie de l'architecture du projet étant déjà dédiée à cela. Un panneau administrateur a été prévu, afin de pouvoir faire de la gestion d'utilisateur, ainsi qu'un panneau d'administration permettant à l'archéologue de modifier le contenu de certaines tables de la base de données pour pouvoir répondre à ses besoins.

Un plan en quatre parties a été adopté pour vous présenter mon travail dans ce stage, qui est donc quasi-exclusivement le développement de l'application. La première partie est consacrée au contexte de ce stage, avec un rappel sur le projet ARCADIA, ses acteurs, ainsi que sur le LIFO qui m'a accueilli. La deuxième partie consistera à définir les besoins exprimés, ce qui était déjà existant et ce qui était attendu. Ce qui amène naturellement vers le troisième point, où une analyse de l'application est délivrée, présentant ce qui a été réalisé et comment les besoins ont été réglés. Pour finir, un point technique sera abordé, permettant de se plonger plus profondément et précisément dans le projet, et de voir comment l'application a été construite, quels problèmes ont été rencontrés, quelles ont été leurs solutions, et quels choix ont été faits.

I. Contexte du stage

1. Les différents intervenants

Il y a trois partenaires différents au sein de ce projet : le LIFO et PRISME (situés sur le campus de l'Université d'Orléans) et le pôle archéologique (situé à la Tour-Neuve dans le centre-ville d'Orléans).

Le LIFO regroupe cinq équipes : CA (Contraintes et Apprentissage), GAMoC (Graphes, Algorithmes et Modèles de Calcul), LMV (Logique, Modélisation et Vérification), PaMDA et SDS (Sécurité et Distribution des Systèmes). Chaque équipe est composée de professeurs, maîtres de conférence, doctorants, … pouvant travailler sur des projets communs ou personnels. Mon projet s'est déroulé dans le cadre du projet ARCADIA, mené dans l'équipe CA par Matthieu Exbrayat (maître de stage) et Lionel Martin.

Le projet ARCADIA est aussi réalisé par PRISME, qui est composé de deux pôles, F2ME (Fluides, Mécanique, Matériaux et Énergétique) et IRAuS (Image, Robotique, Automatique et Signal). C'est ce second pôle qui est en charge du projet, via l'équipe Image-Vision et plus particulièrement Aladine Chetouani et Sylvie Treuillet.

Plusieurs doctorants co-encadrés par le LIFO et PRISME ont également travaillé sur ce projet, comme actuellement Teddy Debroutelle.

Enfin, le pôle d'archéologie d'Orléans, à l'origine du besoin pour lequel ce projet a été créé, est représenté par Sébastien Jesset, de la direction de la planification, de l'aménagement urbain et de l'habitat.

2. Quelle place dans ce projet ?

Ce stage a une place particulière et évolutive dans le projet ARCADIA. Au départ, la finalité était le développement d'un outil collaboratif de recherche, assez général et pouvant être adapté à plusieurs autres projets et développements par la suite. Cependant, au fur et à mesure que le cahier des charges et les demandes utilisateur se précisait, il s'est centré sur l'outil d'aide à la classification, automatique ou non, spécifique au projet ARCADIA et aux besoins des archéologues. Il est possible maintenant de charger une liste de tessons ainsi que leurs spécificités via un fichier CSV, d'enregistrer des tessons à la main, de les modifier ou de les afficher depuis la base de donnée via certains critères de recherche. Une partie Administrateur, pour notamment gérer les utilisateurs, et une section Paramètres destiné aux archéologues pour modifier certains tables de la base de données

ont aussi été réalisés afin que l'application puisse servir de façon relativement autonome aux besoins réels des archéologues.

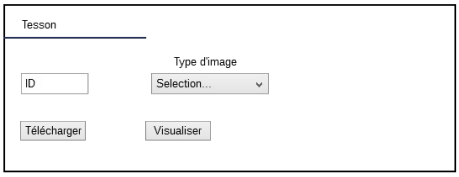
Pour l'instant, l'application peut être vue comme une reprise améliorée de l'outil initial sous FileMaker. Elle intègre de plus tout l'aspect de la classification absent sur le premier outil, que ce soit dans son architecture ou ses fonctionnalités.

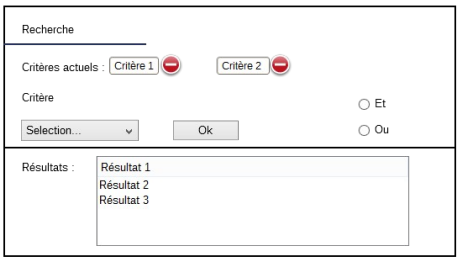
II. LES DÉBUTS DE L'APPLICATION.

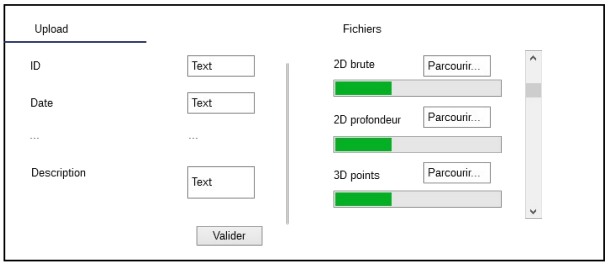
Diagramme de Gantt.

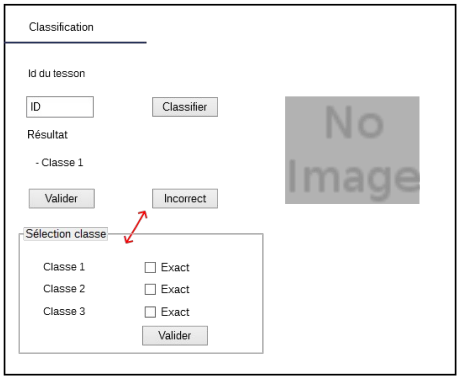
1. La phase de préparation

Pencil.

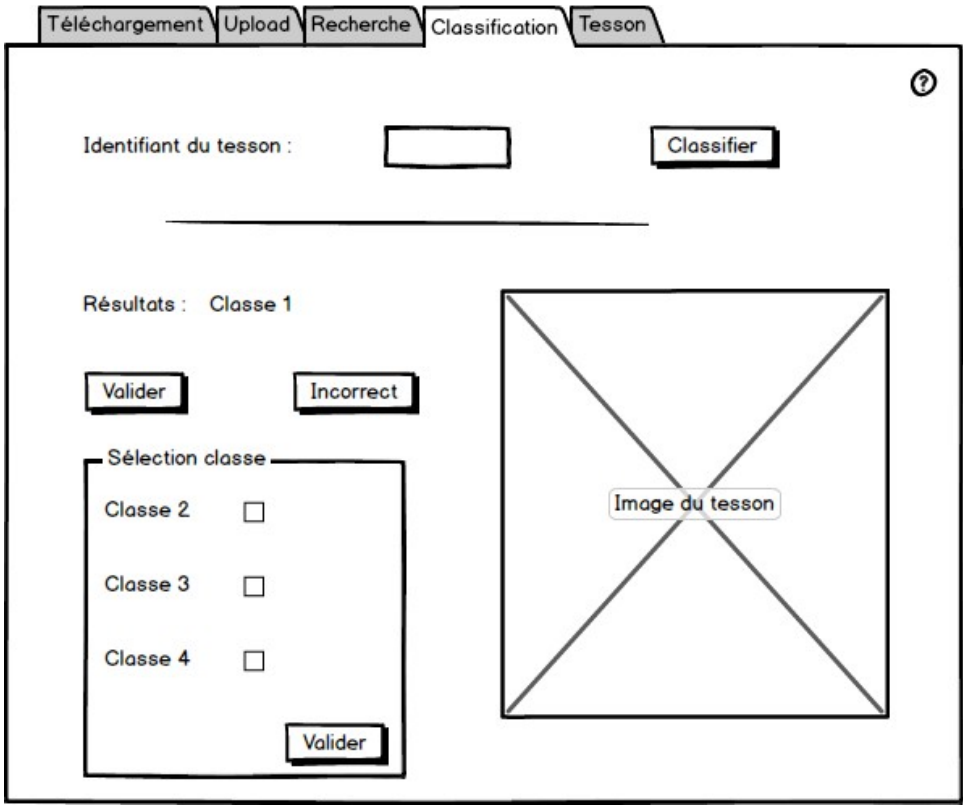








Balsamik.



La première semaine de stage a été consacrée à la préparation du projet. Il s'agissait tout d'abord de faire une synthèse interne des besoins que l'on pensait être attendus. Pour modéliser ces besoins, un *wireframe* (ou *mockups*) a alors été créé afin de faire des premiers écrans et de commencer à réfléchir sur l'architecture que devrait prendre le projet. Deux logiciels ont été utilisés pour ça, tout d'abord Pencil, gratuit mais qui a vite montré quelques limites, puis Balsamiq, payant mais possédant une durée d'essai de un mois largement suffisante dans le cadre de ce stage, plus complet et au design plus travaillé.

Ensuite, le wireframe, accompagné d'un petit dossier explicatif de chaque écran, a été envoyé aux différentes parties du projet ARCADIA, et un rendez-vous avec l'archéologue a été planifié pour réaliser un premier cahier des charges et commencer le développement de l'application. En parallèle, plusieurs recherches ont été faites, d'une part pour savoir si un outil collaboratif de recherche, voire de gestion de workflow existait déjà pour un environnement similaire, et d'autre part pour définir le langage dans lequel l'application allait être codée. La recherche s'est avérée infructueuse, et Symfony fut défini comme framework cible.

2. L'apprentissage du framework

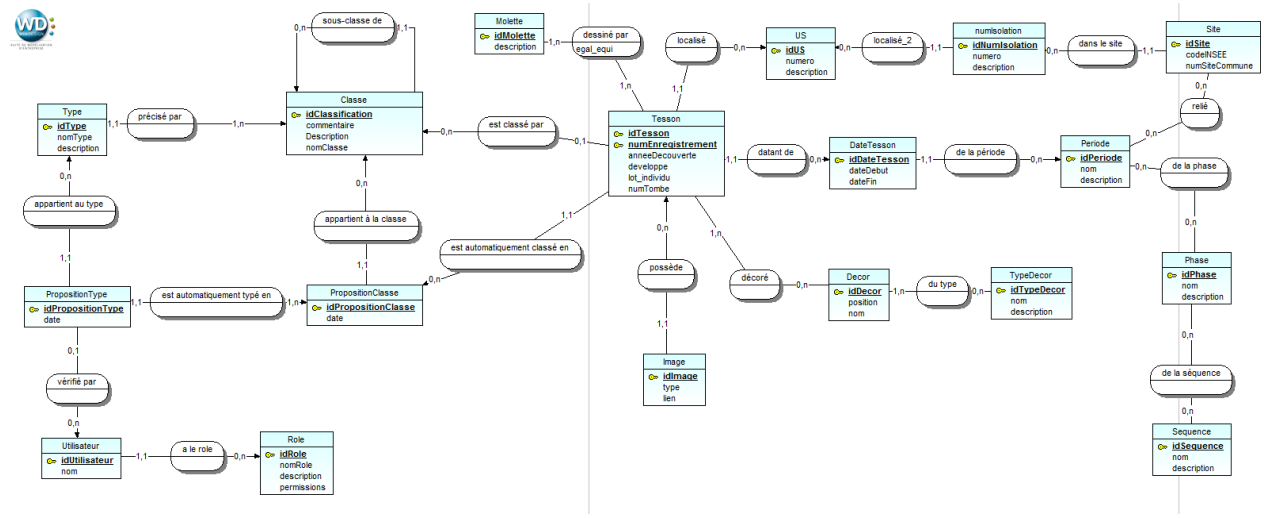
Apprendre les bases de Symfony a été une partie importante, au niveau du temps, dans ce stage. Non seulement c'était la première fois que j'avais à utiliser un framework aussi important que Symfony, mais la façon de penser est très différente de celle du PHP de base. Plus d'interactions directe avec la base de données, de la programmation orientée objet, des entités reliées entre elles en guise de table, ... En plus de cet "esprit" propre à Symfony, il a fallu apprendre son architecture, son arborescence riche, son système de bundles, …

Pour plus d'informations concernant cette partie du rapport, une synthèse sur le fonctionnement de Symfony et sur mon architecture est proposée en annexe.

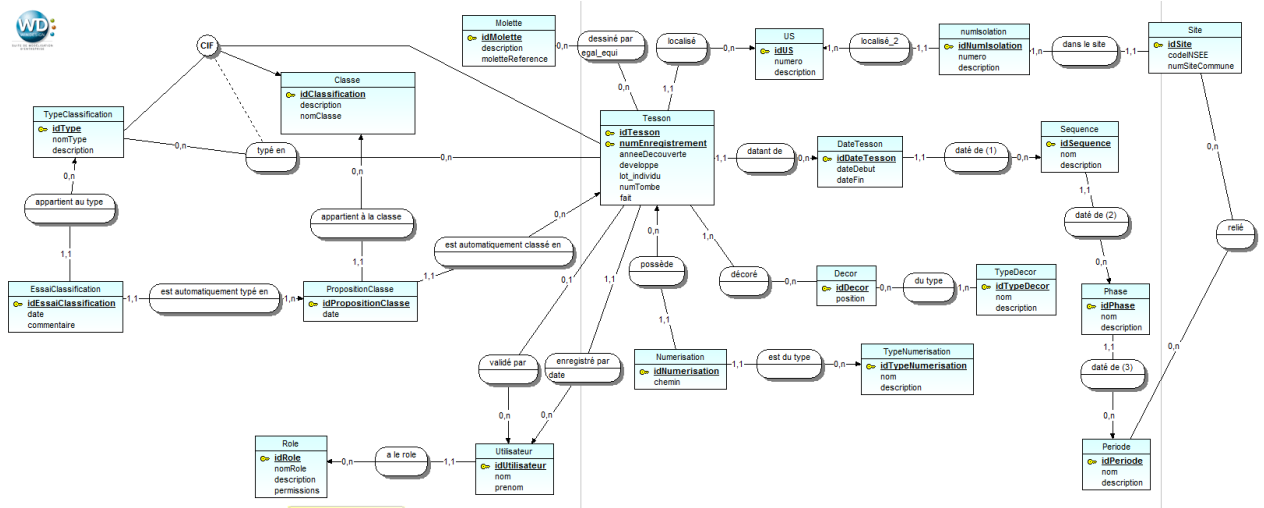
III. ARCHITECTURE

La modélisation de ce projet a été un élément long mais indispensable. Elle a été réalisée via le Modèle Conceptuel de Données (MCD) de Merise, et a beaucoup changé au fil du temps, notamment suivant les rencontres effectuées avec l'archéologue, ou les besoins de développement. Afin d'illustrer cette évolution, voici deux anciennes versions de la modélisation, suivies de la dernière en date.

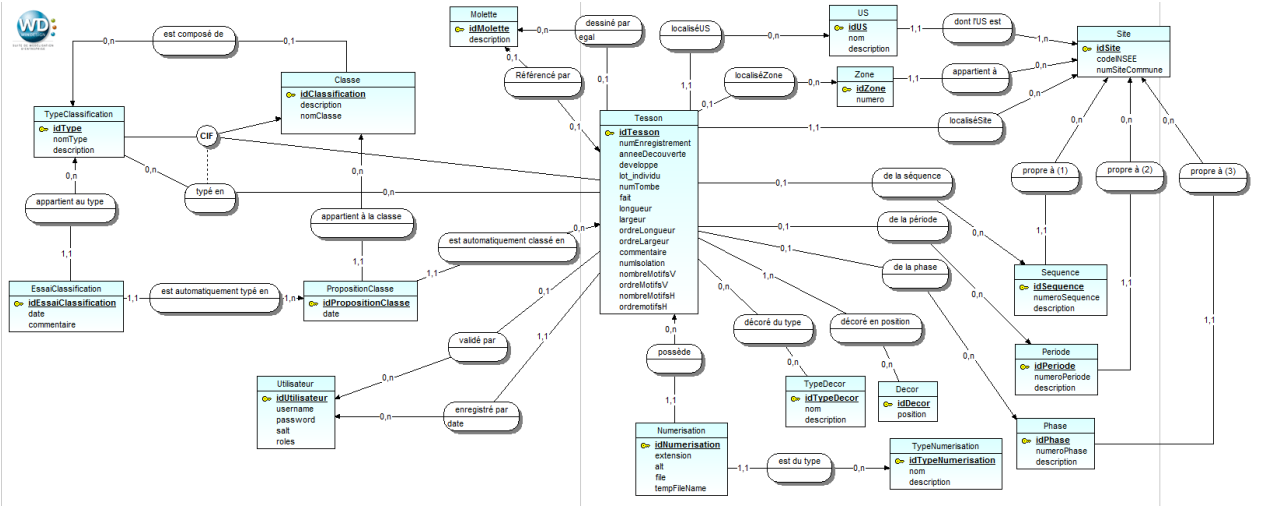
Version 2.0



Version 2.6



Version finale 3.2.4



-Tesson

La classe Tesson est l'élément central de l'architecture, beaucoup d'autres classes s'articulent en fonction d'elle. Elle possède différentes informations, que ce soit des données relatives à la taille ou au contexte de découverte. L'attribut numEnregistrement est présent car c'était l'identifiant d'un tesson dans l'ancienne base de données Filemaker. Il sert uniquement dans l'import CSV provenant de FileMaker pour savoir s'il s'agit d'un nouveau tesson à enregistrer, ou s'il faut simplement mettre à jour ses informations.

-Molette

La classe Molette représente les molettes dessinant les motifs sur les tessons. La relation la reliant à Tesson, *dessiné par,* est modélisée dans la base de donnée par une classe TessonMolette. TessonMolette possède un unique argument, *egal*, qui vaut *true* si la relation entre le tesson et la molette est déclarée "egal", c'est-à-dire s'il y a une correspondance forte avec le répertoire des matrices et qu'on peut donc l'identifier, et *false* si elle est déclarée "equi", c'est-à-dire que l'on se rapproche seulement de ce répertoire. Le lien *Référencé Par,* représente quand à lui si ce tesson est défini en temps que référence pour la molette correspondante. Le lien existe uniquement s'il y a référence, et une molette ne peut être référencée que par un tesson à la fois.

-Localisation

La localisation d'un tesson se fait avec un Site, composé du code INSEE de la commune et du numéro de site, puis d'une US (Unité Stratigraphique). Le site peut également être divisé en plusieurs Zones. En plus de l'id d'un tesson, que ce soit dans l'import ou la saisie directe, on peut retrouver l'unicité en faisant Site + US + numéro d'isolation. Ce dernier est un attribut de tesson assurant l'unicité suivant le couple Site + US, c'est-à-dire que s'il n'est pas précisé, on récupère le plus grand numéro d'isolation pour ce couple, et on assigne au numéro d'isolation du tesson courant ce plus grand nombre, auquel on additionne un.

-Datation

La datation s'effectue suivant trois éléments : la séquence, la phase et la période. Chacune d'entre elles est propre au site. Le MCD possède une ambiguité à ce niveau-là, car autant chaque tesson possède ou non chacune des trois datations, qui sont elles-mêmes propres à un site, autant il pourrait dans cette modélisation y avoir des Séquences non rattachée à la bonne Phase, ou des Phases non rattachées aux bonnes Périodes.

-Numérisation

Un tesson peut avoir ou non des numérisations. Celles-ci sont regroupées par type de numérisations, comme par exemple des nuages de points, des cartes des profondeurs, ... L'attribut alt représente son nom complet, tel qu'il a été uploadé. Extension est le type de fichier (jpg, pgn, …), file et tempfilename servent à l'upload de la numérisation.

-Décors

La classe décor correspond en réalité à la position de décor. Les décors ainsi que le type de décors ont été séparés en deux classes différentes, car un tesson peut avoir différents décors à différents endroits de la poterie originelle.

-Utilisateur

L'utilisateur découle de la classe user de Symfony. L’attribut salt n'est pas utilisé pour hasher les mots de passe, car bcrypt a été utilisé pour crypter les mots de passe. Les rôles sont enregistrés sous forme de tableau php dans la base de données.

-Classification

Un TypeClassification est composé d'une ou plusieurs classes. Par exemple un type E regroupera les classes EA, EB, EC, … La CIF ternaire qui existe entre Tesson, Classe et TypeClassification, appelée TypageEn dans la structure, représente la contrainte suivante : S'il y a typage, alors celui-ci concerne à la fois un tesson et un type de classification par rapport à une classe. De plus, une proposition de classe et une proposition d'un type de classification sont envisagés dans le cadre futur d'une classification automatique d'un tesson.

IV. TRAVAIL RÉALISÉ.

1. Gestion des utilisateurs, rôles et sécurité.

Les utilisateurs de l'application découlent de l'interface *user* de Symfony. Une gestion complète de ces utilisateurs est prévue dans le panneau d'administration par ceux ayant le statut d'administrateur. La sécurité des utilisateurs, que ce soit au niveau des rôles ou des mots de passe, est définie dans app/config/security.yml. La sécurité se passe à trois niveaux différents :

- ✗ Dans app/security des URLs sont entièrement sécurisées. Par exemple pour accéder à admin/…, il faut obligatoirement avoir le rôle admin.
- ✗ Au niveau des controllers, chaque fonction est protégée par des annotations, ainsi uniquement les archéologues peuvent accéder aux parties leur étant destinées, idem pour les administrateurs.
- ✗ Pour la vue, les templates Twig ne font voir à chaque utilisateur que ce qu'il est censé voir, ainsi un utilisateur normal ne verra pas d'onglet Paramètres ou Administration.

Tous les mots de passe de la base de données sont encryptés par la fonction de hashage *bcrypt*, qui permet d'enregistrer deux mots de passes identiques différemment et qui est donc très sécurisée. Une authentification est bien évidemment nécessaire pour se connecter à l'application. Elle redirige vers la page d'upload d'un nouveau tesson par défaut, sauf si l'utilisateur vient d'une autre page de l'application, auquel cas il sera redirigé vers celle-là. La seule page non sécurisée de l'application est le formulaire de connexion. Ensuite, Symfony fait le travail en interne, en redirigeant la requête de l'utilisateur de façon sécurisée pour l'authentifier ou non.

Les rôles sont également définis dans app/config/security.yml. Il en existe pour l'instant trois : ROLE_USER pour les utilisateurs lambdas de l'application, ROLE_ARCHEOLOGUE permettant d'accéder aux paramètres et ROLE_ADMIN pour pouvoir voir le panneau d'administration et donc de gestion des utilisateurs. Ils sont classés dans cet ordre : ROLE_USER < ROLE_ARCHEOLOGUE < ROLE_ADMIN, ce qui signifie qu'un archéologue peut effectuer les actions d'un utilisateur standard mais pas l'inverse.

Un utilisateur par défaut a été défini, Import_CSV, pour permettre l'importation massive de tessons via un fichier CSV, car chaque tesson doit être relié à un utilisateur. Il paraît judicieux de le supprimer une fois l'application déployée pour éviter d'éventuel problèmes de sécurité.

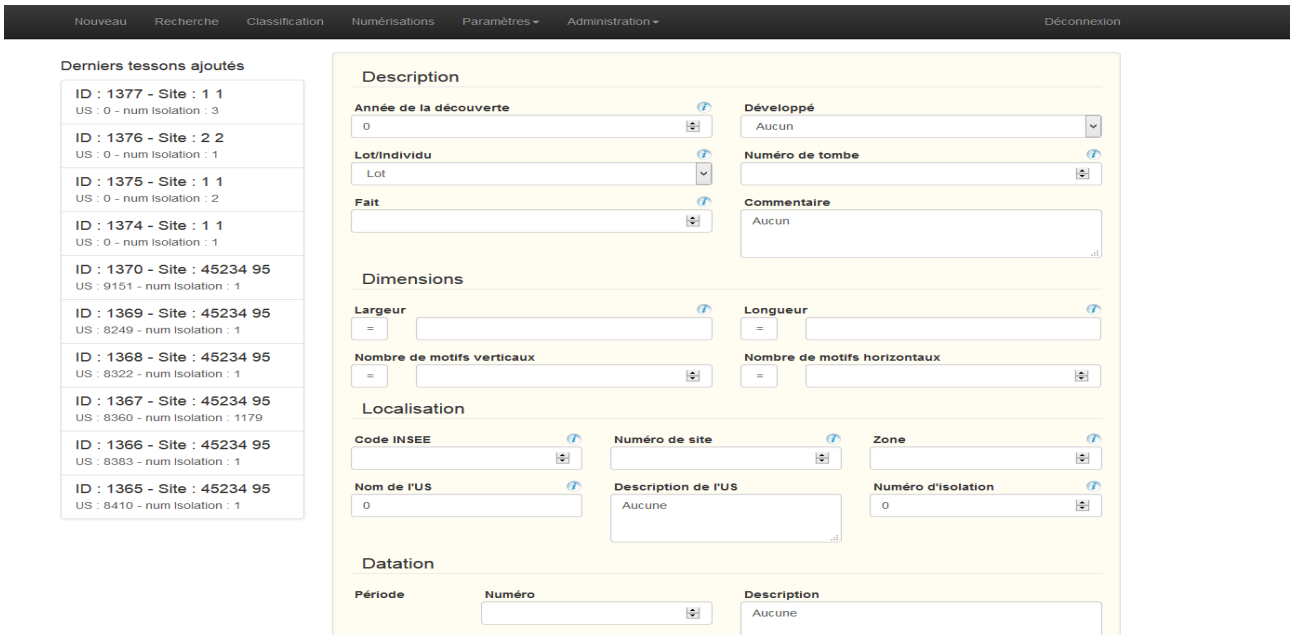
2. Import CSV

L'import CSV est du code destiné à n'être utilisé qu'un seule fois, au début du lancement de l'application pour remplir la base de données. Sous forme de service, il s'exécute en console et parse un fichier CSV encodé en utf8 pour enregistrer ses informations en base de données. Le tableau CSV est convertit par une fonction annexe (Services/ConvertCsvToArray) en un tableau PHP afin de faciliter la manipulation de ce dernier. L'objet ProgressBar est utilisé afin d'avoir une barre d'avancement dans la console permettant d'avoir une visualisation du nombre de tessons enregistrés et restants. Il fonctionne selon certains principes afin de garder une base de données cohérente et propre, différents de l'enregistrement classique d'un tesson dans l'application. Il faut en effet vérifier très minutieusement chaque cellule afin d'être sûr de ne pas avoir d'erreurs. De plus, afin que le service marche, la première ligne de chaque colonne doit être nommée selon une façon très précise. Ce côté strict du code n'a pas été corrigé du fait de son utilisation : comme il n'est destiné qu'à servir une fois puis être laissé de côté, il est réalisé de façon un peu moins intuitive que le reste, ce qui peut être malheureusement un frein à son utilisation si elle n'est pas précise.

Fichier CSV importé.

	J	K	L	M	N	O	P	Q	R	S	T
1	largeur	longueur	LOT	molette référence	N° enregistrement	N° isolation 1	N° PERIODE	N° PHASE	N° SEQUENCE	nbre motifs h	nbre motifs v
2	0,7	>3,5	LOT		1087		9.2	1377	3105	>11	3
3	0,7	>3,5			1088		9.2	1377	3105	>11	3
4	0,7	>3,5			1089		9.2	1377	3105	>16	2
5	>0,6	>1,7			1143		9.2	1378	3100	>7	>2
6	1	4,1		non	1196	5034-1	7.1	1359	3166	12	4
7	0,7	>2,1			1213		9.2	1396	3355	>8	2
8	0,8	>2,2			1217		7.1	1361	3158	>6	4
9					1237		10.1	1382	3096		
10	0,8	4,3		oui	1310	5099-2	7.1	1400	3193	10	2
11	1,6	>3,2			1370		7.2	242	174	>6	3
12	>1,2	4,45		non	1402	2179-1	7.2	242	174	18	3 ou 5(?)
13	0,8	>3,1		oui	1475	2197-3	7.1	240	178	>5	2
14	1,3	>4,2			1496	2194-2	7.2	243	172	>4	1
15	1,3	inconnue			1497		7.2	243	172	inconnu	1
16	1,15	>2,9			1516		8	245	168	>8	2
17	1,00	>3,5			1524		8	245	168	>5	3
18					1529		?	?	?		
19	>0,3	>=4,9		oui	1571	2084-3	7.1	235	189	>=12	>=1
20	>0,7	>3,5		non	1572	2084-4	7.1	235	189	>3	1
21	1,1	>2		non	1573	2084-5	7.1	235	189	>2	1
22	0,7	>2,3		non	1574	2084-6	7.1	235	189	>5	1
23					1584		8	246	166		
24	1,00	>3,6			1588		8	246	166	>3	1
25	0,9	3,20		oui	1604	3062-1	7.1	329	680	19	3
26	0,95	>3,6		non	1605	3062-4	7.1	329	680	>15	2
27	0,7	>2,6		non	1606	3062-3	7.1	329	680	>7	3
28	0,9	>3,4		oui	1608	3062-2	7.1	329	680	>12	8
29	?	>4			1626		7.2	332	672	>3	1 ou 2
30	0,7	5,7		oui	1627	3050-1	7.2	332	672	15	2
31	0,6	>5		non	1628	3050-2	7.2	332	672	>15	1

3. Nouveau



a. Enregistrement et modification

L'onglet Nouveau, auparavant appelé Upload, est destiné à la saisie et l'enregistrement d'un tesson dans la base de données. Une partie du code a été récupérée du service Import CSV, car certains aspects sont relativement proches. Techniquement parlant, plusieurs entités sont créées dès le début du controller, afin de pouvoir afficher leurs valeurs par défaut dans la vue Twig du formulaire. La même action se déroule dans le cas d'une modification, mais au lieu d'afficher les valeurs par défaut des différentes entités, il affiche les valeurs existantes du tesson courant.

```
36
37      /**
38       * @Security("has_role('ROLE_USER')")
39       */
40      public function uploadAction(Request $request) {
41
42          $tesson = new Tesson ();
43          $tesson->setUS ( new US () );
44          $tesson->setSite ( new Site () );
45          $tesson->setPeriode ( new Periode () );
46          $tesson->setPhase ( new Phase () );
47          $tesson->setSequence ( new Sequence () );
48          $tesson->getUS ()->setSite ( $tesson->getSite () );
49          $utilisateur = $this->getUser();
50
51          $tesson->setEnregistrePar ( $utilisateur );
52          $form = $this->get ( 'form.factory' )->create ( TessonType::class, $tesson );
53
54          if ($request->isMethod ( 'POST' ) && $form->handleRequest ( $request )->isValid ()) {
55              $this->verifierFormTesson($tesson);
56
57              $request->getSession ()->getFlashBag ()->add ( 'notice', 'Tesson enregistré' );
58              return $this->redirectToRoute ( 'lifo_classif_tesson', array ('id' => $tesson->getId ()) );
59          }
60
61          return $this->render ( 'LIFOClassifBundle:Platform:upload.html.twig', array (
62                  'form' => $form->createView ()
63          ) );
64      }
65
```

La seule chose réellement faite dans la méthode uploadAction(), à part cette initialisation et l'appel de la vue, consiste à créer le formulaire. Celui-ci hydrate l'entité Tesson (en Symfony, quand un formulaire est relié à une entité, on dit qu'il l'hydrate), qui appelle la création de nombreux autres formulaires annexes reliés à d'autres entités. Différents types de présentation de ces entités annexes ont été réalisées pour ce formulaire. Par exemple, Decor et TypeDecor sont uniquement listées sous forme de checkbox par rapport à un de leurs attributs, Période, Phase et Séquence ont leurs propres formulaires qui sont appelés pour compléter celui de Tesson, ou encore Numerisation est sous forme de collection. Tous ces formulaires sont mis bout à bout et assemblés par le FormBuilder de Symfony, afin de créer un unique formulaire. Il est ensuite envoyé à la vue, qui le découpe en différents morceaux afin de construire une page HTML propre, et stylisée grâce au Bootstrap présent nativement dans Symfony.

```
79              ->add('US', USType::class)
80              ->add('zone', ZoneType::class)
81              ->add('site', SiteType::class)
82              ->add('numIsolation', IntegerType::class)
83              ->add('periode', PeriodeType::class, array(
84                  'required'     => false))
85              ->add('phase', PhaseType::class, array(
86                  'required'     => false))
87              ->add('sequence', SequenceType::class, array(
88                  'required'     => false))
89              ->add('decor', EntityType::class, array(
90                  'class'         => 'LIFOClassifBundle:Decor',
91                  'choice_label'  => 'position',
92                  'multiple'      => true,
93                  'expanded'      => true
94              ))
95              ->add('typeDecor', EntityType::class, array(
96                  'class'         => 'LIFOClassifBundle:TypeDecor',
97                  'choice_label'  => 'nom',
98                  'multiple'      => true,
99                  'expanded'      => true
100             ))
101             ->add('numerisation', CollectionType::class, array(
102                 'entry_type'   =>NumerisationType::class,
103                 'allow_add'    => true,
104                 'allow_delete' => true,
105                 'required'     => false
106             ))
107             ->add('tessonMolette', TessonMoletteType::class)
108             ->add('enregistrer', SubmitType::class)
109         ;
```

L'affichage de Numerisation est particulier, car un script permet d'ajouter ou d'enlever à loisir des numérisations, pour pouvoir en ajouter autant que nécessaire dans la base de données. Quand à l'enregistrement d'une numérisation, il est tout aussi particulier car c'est le seul endroit où l'utilisateur peut enregistrer sur le serveur des numérisations venant de sa machine. Grâce aux *lifecycle callbacks*, certaines méthodes sont automatiquement appelées après certaines actions sur cette entité, suivant les différentes étapes de sa vie. Par exemple, cette fonction de préupload est appelée à deux moments de la vie de l'entité, soit quand elle est crée, soit quand elles est mise à jour, grâce aux annotations au-dessus de la méthode.

```php
    /**
     * @ORM\PrePersist()
     * @ORM\PreUpdate()
     */
    public function preUpload() {
        if (null === $this->file) {
            return;
        }
        $this->extension = $this->file->guessExtension ();
        $this->alt = $this->file->getClientOriginalName ();
    }

    /**
     * @ORM\PostPersist()
     *
     * @ORM\PostUpdate()
     */
    public function upload() {
        if (null === $this->file) {
            return;
        }
        $strAlt = "t" . $this->tesson->getId () . "-n" . $this->id . "." . $this->extension;
        $this->alt = $strAlt;
        if (null !== $this->tempFilename) {
            $oldFile = $this->getUploadRootDir () . '/t' . $this->tesson->getId () . "-n" . $this->id . '.' . $this->tempFilename
            if (file_exists ( $oldFile )) {
                unlink ( $oldFile );
            }
        }
```

La majeure partie du code se trouve dans une fonction annexe, verifierFormTesson($tesson), également utilisée par la modification d'un tesson existant afin d'éviter la duplication de code. Cette méthode vérifie pour chaque entrée de l'utilisateur si une instance de l'entité est déjà présente en base de données. Si elle est déjà présente, elle l'utilise, et sinon elle en crée une nouvelle. Chaque entrée du formulaire est vérifiée pour être sûr que l'utilisateur n'a pas rentré de mauvaise information, même si, avec un navigateur web le supportant, HTML5 empêche déjà de rentrer par exemple des lettres dans un champ destiné à un entier, …

Pour ce qui est des champs pouvant être vides, la première chose faite est de vérifier s'ils sont NULL, afin d'éviter toute lecture de code inutile.

Le cas du numéro d'isolation est à part. L'utilisateur peut le laisser vide. A ce moment-là, pour préserver l'unicité d'un tesson créée par Site + US + numéro d'isolation, la méthode du controller va chercher le plus grand numéro d'isolation dans la base de donnée par rapport au couple US + Site, et lui attribuera ensuite ce numéro +1. Si aucun n'a été trouvé pour ce couple, le numéro attribué sera 1. Il n'est donc pas possible d'avoir des numéros d'isolation ≤ 0.

```php
    public function findNumIsolationMax($us_id, $site_id) {
        $qb = $this->_em->createQueryBuilder('t')
        ->select('MAX(t.numIsolation)')
        ->from('LIFOClassifBundle:Tesson', 't')
        ->leftJoin('t.us', 'u')
        ->leftJoin('t.site', 's')
        ->where('u.id=:us_id')
        ->andWhere('s.id=:site_id')
        ->setParameter('us_id', $us_id)
        ->setParameter('site_id', $site_id);
        return $qb->getQuery()->getSingleScalarResult ();
    }
```

Le code gérant TessonMolette et Molette est lui aussi particulier. La condition de départ ne permet que de poursuivre si egal vaut *true* (egal) ou *false* (équi), et qu'une molette a bien été

renseignée. Si la molette est trouvée en base de données, elle sera prise, sinon une nouvelle sera créée. Si l'utilisateur décide que ce sera une référence, le lien est créé entre Molette et Tesson. Comme ce lien est unique, si un autre Tesson était référence, cet ancien lien sera perdu. La description de la Molette, si renseignée, pourra également être changée.

```
483             if(($tesson->getTessonMolette()->getEgal() == true || $tesson->getTessonMolette()->getEgal() == false)
484                     && $tesson->getTessonMolette()->getMolette()->getNom() != "" ){
485                         $reference=$tesson->getTessonMolette()->getMolette()->getReference();
486                         $descriptionMolette=$tesson->getTessonMolette()->getMolette()->getdescription();
487                         $molette=$em->getRepository('LIFOClassifBundle:Molette')->findOneByNom($tesson->getTessonMolette()->getM
488                         if(is_object($molette)){
489                             $tesson->getTessonMolette()->setMolette($molette);
490                         }
491                         if($reference == true){
492                             $tesson->getTessonMolette()->getMolette()->setReferencePar($tesson);
493                         }
494                         if($descriptionMolette == NULL){
495                             $tesson->getTessonMolette()->getMolette()->setDescription("Aucune");
496                         }
497                         $em->persist($tesson->getTessonMolette()->getMolette());
498                         $em->persist($tesson->getTessonMolette());
499             } else {
500                 $tesson->setTessonMolette(NULL);
501             }
```

Enfin, de multiples bulles d'aides sont disponibles à côté des champs, afin qu'avec un simple survol de l'infobulle avec sa souris l'utilisateur ait des précisions ou des explications sur le champs qu'il s'apprête à compléter.

b. Affichage du récapitulatif d'un tesson

Après l'enregistrement d'un tesson, le clic sur le menu contextuel ou une recherche, la page récapitulative d'un tesson et de ses spécificités est affichée. Elle utilise la mise en forme des tableaux Bootstrap. Les caractéristiques ont été regroupées dans différents tableaux suivant le type d'informations qu'elles donnent. Le caractère moyen (un champs permettant à l'archéologue de se faire rapidement une idée de la taille du tesson) est calculé directement, il n'est pas enregistré. Il est précédé d'un test qui vérifie qu'aucune des quatre variables suivantes n'est vide (auquel cas il affichera 0), puis donne le résultat de l'opération suivante : (longueur * largeur) / (nombre de caractères verticaux * nombre de caractères horizontaux).

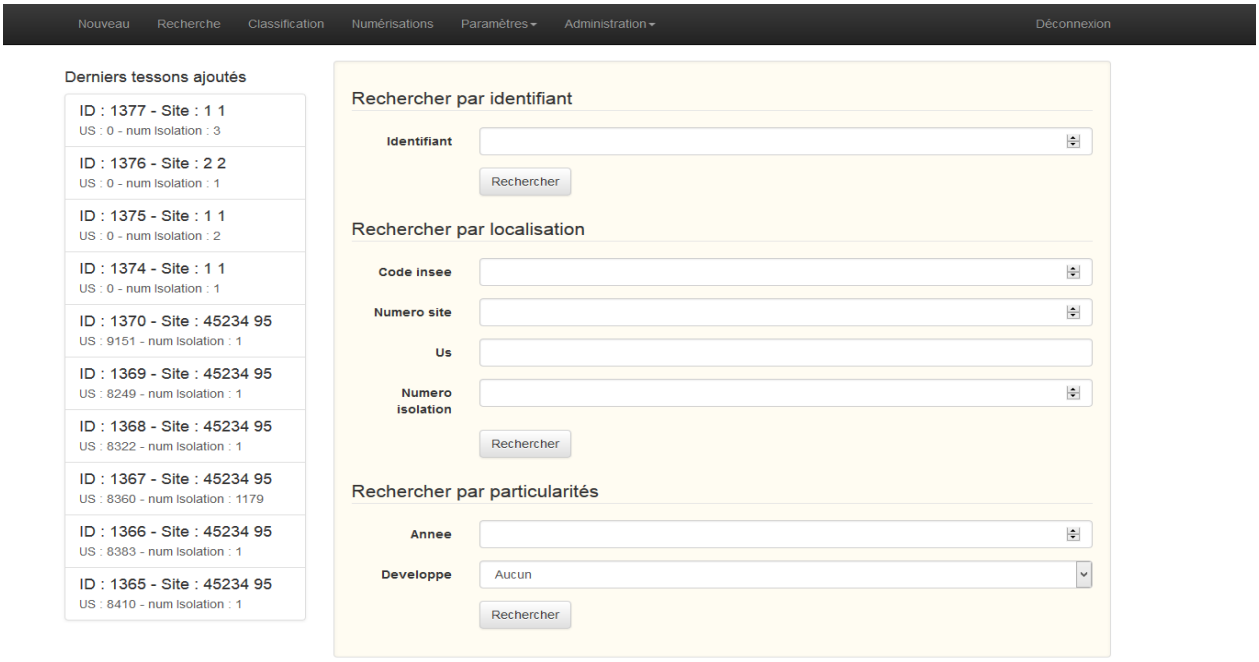
Un bouton "Modifier" en haut de la page permet directement à l'utilisateur de modifier un tesson s'il trouve une erreur dans la fiche.

Si des éléments sont manquants, une valeur par défaut est affichée : "Non renseigné" afin d'avoir toujours le même type de récapitulatif.

Les numérisations sont classées par type, et ne sont redimensionnées que sur l'axe horizontal. Ainsi, leur place dans le tableau ne dérange pas sa taille horizontale qui sera toujours la même, et l'axe vertical étant automatiquement redimensionné, il n'y a pas de déformation des numérisations qui aurait été engendré par un redimensionnement des deux axes.

Le code du controller de cette partie est quasiment vide, tout l'affichage se faisant via les templates Twig associés à la vue.

4. Recherche



La recherche par identifiant fonctionne de manière relativement simple, il suffit de taper l'ID du tesson pour qu'une requête soit envoyée à la base de données pour récupérer le tesson correspondant et l'afficher via son récapitulatif. Si aucun tesson n'est trouvé, un message d'erreur apparaît.

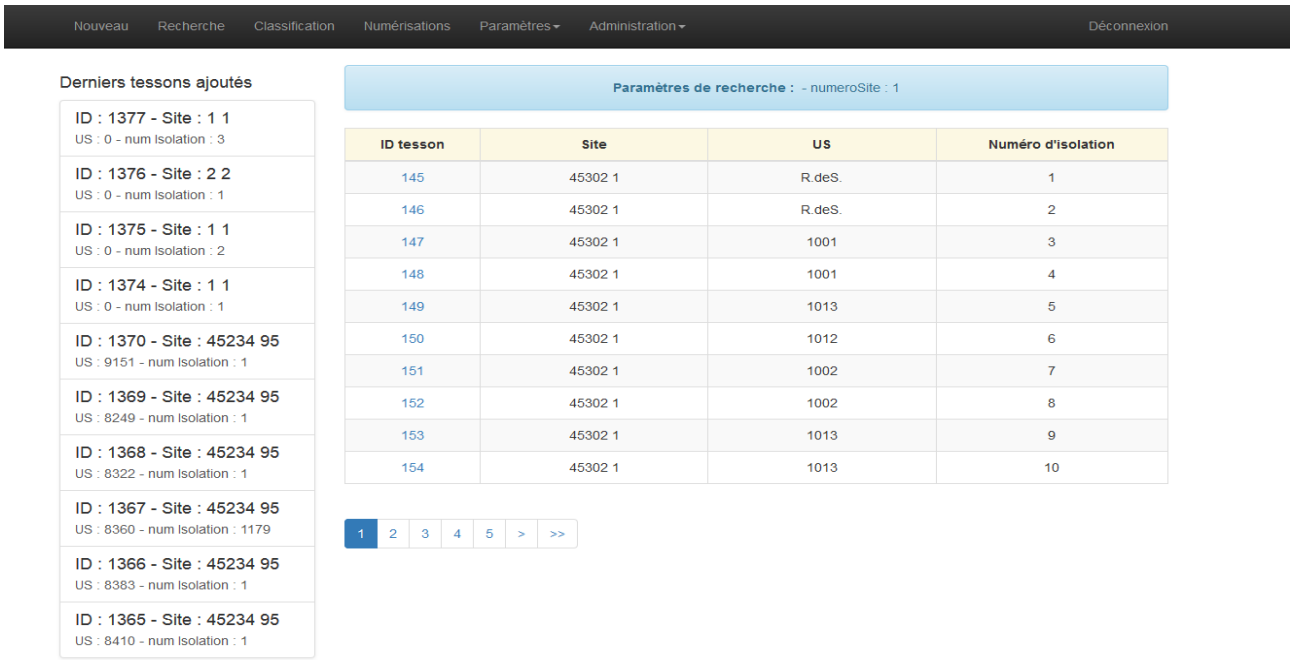
La recherche par localisation fonctionne de manière plus complexe, et possède quatre champs : code INSEE, numéro de site, US et numéro d'isolation. Au départ, elle suivait le même principe que celle par identifiant, affichant le tesson correspondant aux quatre champs remplis, et un message d'erreur apparaissant sinon. Elle a par la suite été améliorée, afin de ne rentrer qu'un ou plusieurs sous-ensembles de recherche, affichant ensuite un tableau listant tous les tessons correspondants à la recherche. On est alors redirigé vers une nouvelle méthode du controller, rechercheAfficherAction(), qui appelle ensuite le repository de tesson pour construire une requête personnalisée afin de gagner en complexité.

```php
        $qb = $this->createQueryBuilder('t')
        ->orderBy('t.id', 'ASC');
        if($criteres['codeINSEE'] != ""){
            $qb->leftJoin('t.site', 's1')
            ->andWhere('s1.codeINSEE=:codeInsee')
            ->setParameter('codeInsee', $criteres['codeINSEE']);
        }
        if($criteres['numeroSite'] != ""){
            $qb->leftJoin('t.site', 's2')
            ->andWhere('s2.numSiteCommune=:numSite')
            ->setParameter('numSite', $criteres['numeroSite']);
        }
        if($criteres['us'] != ""){
            $qb->leftJoin('t.us', 'u')
            ->andWhere('u.nom=:us')
            ->setParameter('us', $criteres['us']);
        }
        if($criteres['numeroIsolation'] != ""){
            $qb->andWhere('t.numIsolation=:numIsolation')
            ->setParameter('numIsolation', $criteres['numeroIsolation']);
        }
        if($criteres['annee'] != ""){
            $qb->andWhere('t.annee=:annee')
            ->setParameter('annee', $criteres['annee']);
        }
        if($criteres['developpe'] != ""){
            $qb->andWhere('t.developpe=:developpe')
            ->setParameter('developpe', $criteres['developpe']);
        }

        $query = $qb->getQuery();

        $premierResultat = ($page - 1) * $nbMaxParPage;
        $query->setFirstResult($premierResultat)->setMaxResults($nbMaxParPage);
        $paginator = new Paginator($query);
```

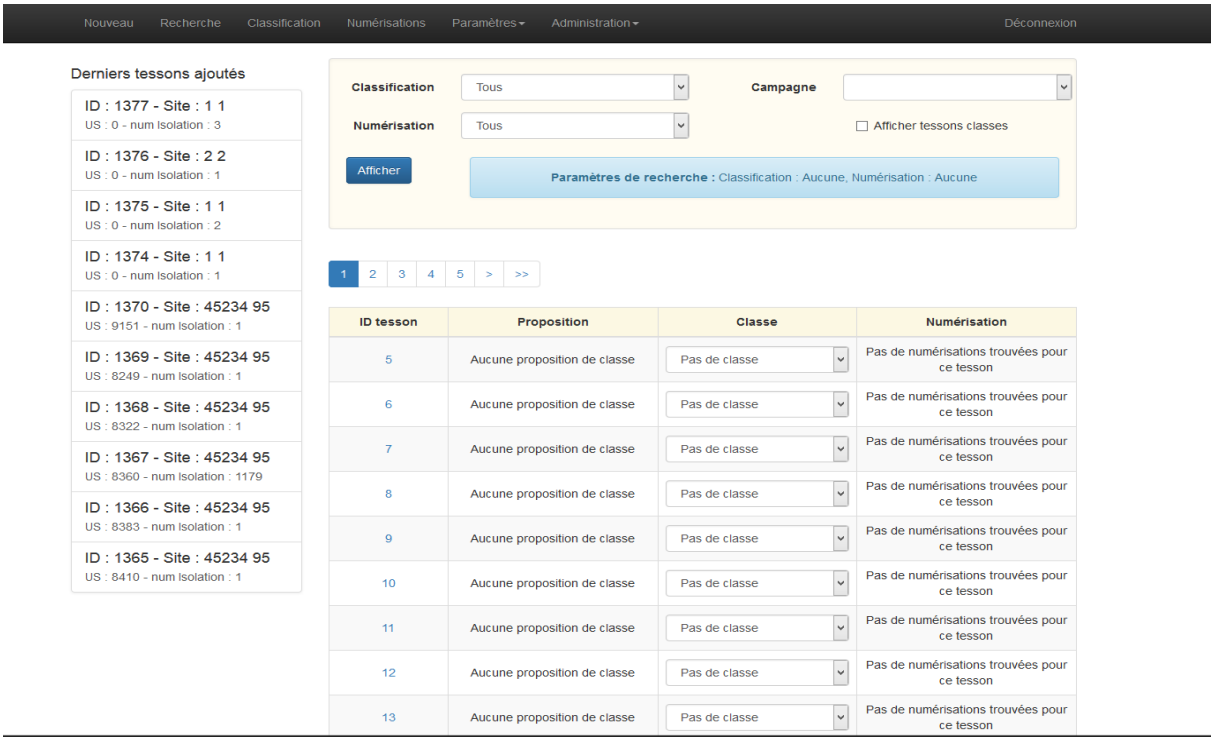
Grâce au QueryBuilder qui permet de fabriquer une requête de façon simple et empirique, chaque champs renseigné est rajouté à la requête, qu'on sort sous forme d'objet Paginator afin d'avoir une pagination des résultats affichés. On ne va ainsi chercher dans la base de données que ce qui nous intéresse exactement, puis on l'affiche sous forme d'un tableau. La première colonne de ce tableau, correspondant à l'ID du tesson, redirige vers sa fiche récapitulative. Les autres colonnes décrivent son emplacement pour que l'utilisateur retrouve facilement le ou les tessons recherchés.



La recherche par particularités fonctionne exactement de la même manière. Comme c'est un tableau de critères qui est envoyé au repository pour construire la requête, elle utilise la même méthode que la recherche par localisation.

Les paramètres de recherche sont affichés en haut du tableau pour permettre à l'utilisateur de voir quels paramètres ont été utilisés.

5. Classification



a. La recherche

La recherche permet de filtrer les résultats affichés dans le tableau suivant certains critères. Comme ceux-ci peuvent être multiples, le controller regarde en premier lieu quels paramètres de recherche sont actuellement appliqués. Il peut y avoir le type de classification, qui n'affichera que les classes inhérentes à ce type et le type de numérisation, qui n'affichera que les tessons possédant des numérisations de ce type. Une checkbox permet à l'utilisateur de choisir d'afficher les tessons déjà classés, qui est cochée automatiquement s'il sélectionne une valeur dans la combo "Classification". La liste déroulante "Campagne" n'est pour l'instant pas utilisée, elle servira plus tard dans le cadre de classification automatique à partir de l'application, afin de ne sélectionner que les tessons appartenant à telle ou telle campagne de classification.

Ensuite, une fois que la méthode a contrôlé les différents critères de recherche, elle construit

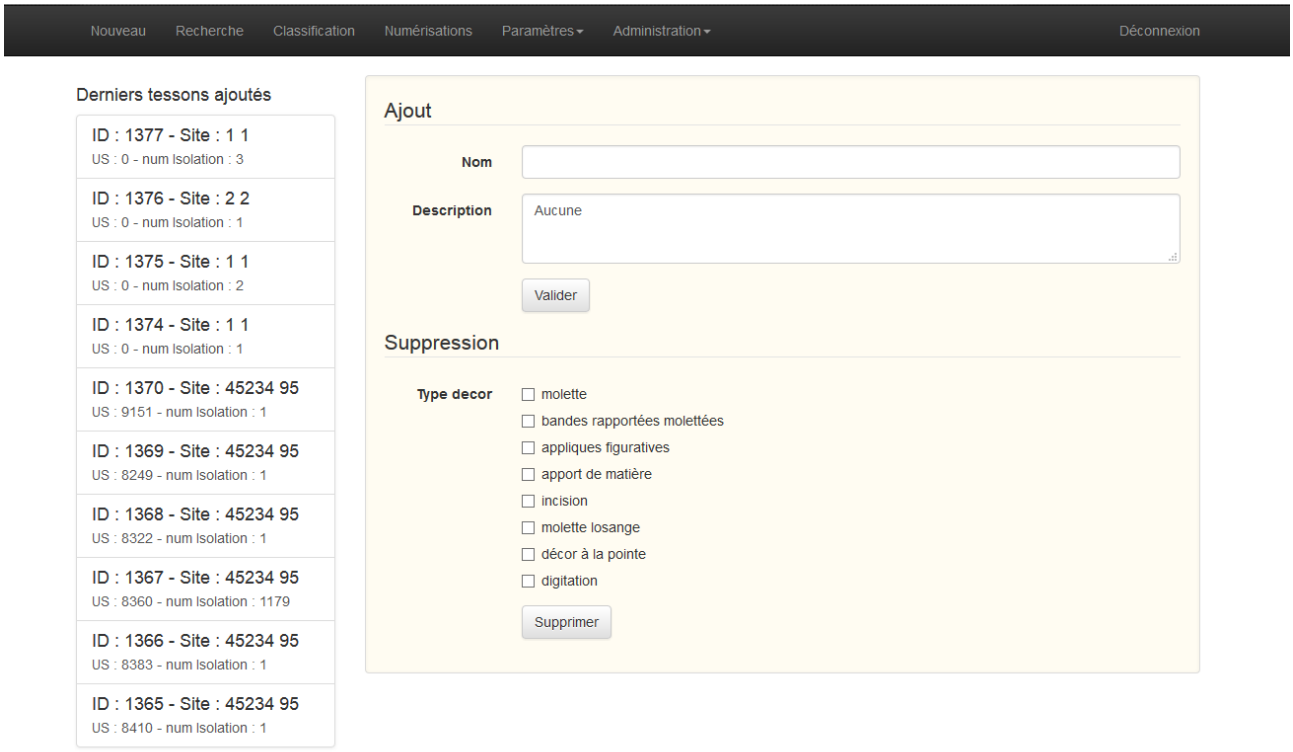
une requête personnalisée via le repository de Tesson de la même manière que la recherche par localisation de l'onglet Recherche. Un message apparaît également pour rappeler à l'utilisateur les paramètres de recherche actuels.

b. Le tableau

L'affichage de la liste paginée des tessons correspondants aux paramètres de recherche se fait sous la forme de tableau.

La première colonne, l'id du tesson, renvoie vers la fiche récapitulative de ce tesson. La colonne proposition, toujours vide pour l'instant, est reliée dans la base de données à la classe PropositionClasse, permettant dans l'utilisation future de l'application pour de la classification automatique de voir quelles classes l'algorithme propose, permettant à l'archéologue dans la colonne suivante d'infirmer ou confirmer ces propositions. La liste déroulante de cette troisième colonne, "Classe", permet donc de modifier directement la classe d'un tesson. Un simple changement de sélection par l'utilisateur envoie une requête changeant la classe. La dernière colonne est dédiée au numérisations. Une miniature y est affichée, mais l'image peut être agrandie par un simple clic. Par défaut, la première numérisation trouvée y est mise, mais si l'utilisateur a sélectionné dans la recherche un type de numérisation particulier, la numérisation de ce type est affichée à la place.

6. Paramètres



Les paramètres étaient auparavant reliés au panneau d'administration. Ils ont été par la suite détachés, suite à la définition du rôle d'archéologue parmi les utilisateurs en plus de celui d'administrateur. Elle permet à l'archéologue de maintenir la base de données à jour. Pour l'instant, seuls les positions et les types de décor sont modifiables, mais cela pourra être modifié par la suite suivant les besoins réels des utilisateurs. L'utilisation de *dropddown* de Bootstrap permet d'afficher une liste déroulante dans l'onglet, permettant de choisir quelle partie de la base de données on désire modifier.

Pour chaque classe modifiable, deux actions sont possibles, la suppression et l'ajout. Deux formulaires spécifiques sont présents dans le controller pour bien différencier les deux. Une spécificité de Symfony fait que l'on peut changer le nom du formulaire uniquement si celui-ci a été créé à partir d'une entité, or, comme c'est un formulaire de recherche, il a été construit par le FormBuilder directement dans le controller.

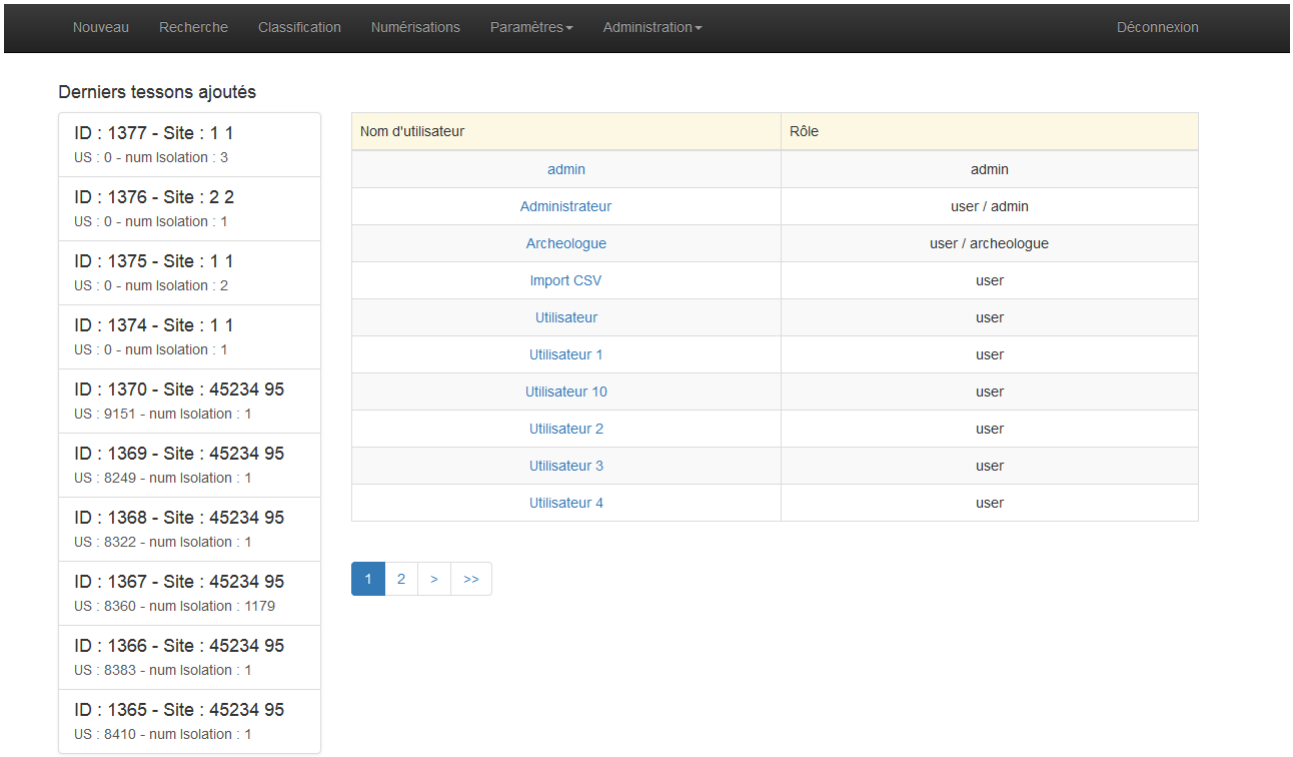
```
4  <div class="well">
5      <fieldset>
6          <legend>Ajout</legend>
7              {{ form_start(formAjoutDecor, {'name':'formAjoutDecor' })  }}
8              {{ form_rest(formAjoutDecor) }}
9              {{ form_end(formAjoutDecor) }}
10     </fieldset>
11     <fieldset>
12         <legend>Suppression</legend>
13             {{ form_start(formSuppressionDecor, {'name':'formSuppressionDecor' })  }}
14             {{ form_rest(formSuppressionDecor) }}
15             {{ form_end(formSuppressionDecor) }}
16     </fieldset>
17 </div>
```

Des problèmes pouvaient donc survenir lors de l'affichage avec deux formulaires aux mêmes noms. Ce problème a été contourné dans la vue Twig.

Grâce aux templates Twig, le formulaire peut être découpé en trois parties : form_start, form_rest et form_end. Les propriétés du formulaires étant dans form_start, il est alors possible de changer son nom, et donc d'avoir sur le rendu HTML deux formulaires aux noms différents, et donc sans conflits possibles.

L'ajout permet simplement d'ajouter une nouvelle entité dans la base de données en renseignant ses différents attributs, tandis que la suppression, affichée sous forme de checkbox, permet la suppression d'une ou plusieurs entités.

7. Administration



Le panneau d'administration ne permet pour l'instant de ne manipuler que les utilisateurs, mais il pourra être étendu par la suite à d'autres fonctions si le besoin se présente. La *dropdown* de Bootstrap est utilisée de la même manière que pour "Paramètres", à l'exception de séparateur de champs qu'il y a en plus afin de séparer la partie Utilisateur des parties futures.

Ajouter un utilisateur fonctionne de manière très simple, avec un formulaire sécurisé pour l'enregistrement du nom d'utilisateur et de son mot de passe (crypté) en base de données. On peut également lui affecter un rôle, celui par défaut étant l'utilisateur standard(ROLE_USER).

Une fonction de recherche permet de rechercher un utilisateur par son pseudonyme. La requête cherche le nom exact donné dans la base de données, s'il y a une faute d'un ou deux caractères dans le champ, l'utilisateur ne sera donc pas trouvé.

On peut également afficher la liste de tous les utilisateurs pour avoir un tableau récapitulatifs des différents comptes enregistrés, avec leur nom et leur rôle. Un clic sur le nom amène vers la fiche de l'utilisateur, où l'on peut alors modifier son rôle.

8. Layout général

L'habillage du site a été réalisé avec Bootstrap. Tous les fichiers annexes nécessaires au fonctionnement du site (Javascript, JQuery, Bootstrap, …) sont intégrés dans l'application. En effet, l'utilisation de CDN (Content Delivery Network, serveurs qui mettent à disposition du contenu de façon très rapide), même si elle est peu coûteuse, peut afficher un site sans habillage ou scripts lors d'un accès sans Internet et/ou dans le cas où l'utilisateur ait vidé son cache. Tout a été enregistré dans le répertoire web de l'application, et des asset ont été utilisés afin de n'avoir normalement aucun problème lors du déploiement du site.

Les templates (méthode utilisée pour afficher de manière quasi-automatisée et similaire des pages web) utilisés dans l'application fonctionnent de la manière la plus propre, avec le modèle à triple héritage : un layout général contenant la structure du site, un layout du bundle contenant les spécificités d'affichage du bundle (quasiment vide dans ce projet) et des templates de pages permettant à la vue d'afficher les bonnes pages du site.

La première partie du layout général est consacrée aux différents imports, comme Bootstrap, d'autres fichiers css, Jquery, ...

La deuxième et majeure partie du layout comporte elle la mise en forme du menu horizontal du haut de la page. Elle y affiche les onglets ainsi que leurs routes correspondantes, et comporte les divers éléments de son affichage, comme les *dropdown*, les restrictions d'accès suivant le rôle, …

Grâce à l'héritage de templates de Twig, ce layout est importé dans chaque fichier html.twig, afin d'avoir une mise en forme de la page toujours identique et sans duplication de code.

9. Aspects secondaires

a. Menu

Le menu tient une place particulière dans l'application du fait de son originalité : ce n'est pas un simple affichage, mais une inclusion de controller dans le layout général, permis par Twig. Ainsi,

à chaque rechargement de page ou redirection, ce menu sera toujours à jour. Il affiche la liste des dix derniers tessons enregistrés. Ainsi, si la personne qui vient de rentrer un tesson se rend compte d'une erreur dans un champs (mal renseigner un champs est une situation très probable après la saisie de plusieurs centaines voire milliers de tessons par l'utilisateur), plutôt que de perdre du temps à rechercher le tesson via l'onglet Recherche, il lui suffira de cliquer sur le tesson correspondant dans le menu. Afin de faciliter la reconnaissance des tessons, uniquement quelques informations essentielles sont affichées dans ce menu : son id et sa localisation précise.

Cet affichage se fait très simplement dans Twig grâce aux boucles.

```
1  <div class="list-group">
2      {% for tesson in listeTessons %} <a
3          href="{{ path('lifo_classif_tesson', {'id': tesson.id}) }}"
4          class="list-group-item">
5          <h4 class="list-group-item-heading">ID : {{ tesson.id }} - Site :
6              {{tesson.site.codeINSEE}} {{tesson.site.numSiteCommune}}</h4>
7          <p class="list-group-item-text">US : {{tesson.us.nom}} - num Isolation
8              : {{ tesson.numIsolation }}</p>
9      </a> {% endfor %}
10 </div>
```

b. Pagination

La pagination aux différents endroits de l'application se découpe en deux phases : le calcul de la requête avec l'objet Paginator, et l'affichage dans la vue.

Dans les repositories, il est possible de retourner une instance de Paginator à la place des résultats de la requête, ce qui est très utile. Dès lors, Symfony s'occupe de cette instance, et il ne nous reste plus qu'à bien la paramétrer et à l'utiliser correctement. Dans le Repository, l'utilisation est très transparente, il suffit de donner à la requête le premier résultat à avoir, obtenu avec l'opération ligne 110, et à lui définir le nombre de résultats maximum.

L'affichage Twig de la pagination a été séparé des autres pages, afin de pouvoir être appelé de n'importe quelle vue. Il suffit alors juste d'inclure pagination.html.twig à l'endroit voulu pour se retrouver avec l'affichage désiré. La duplication de code est ainsi évitée, et l'harmonie de l'application respectée en affichant à chaque fois la pagination de la même façon.

V. Problèmes spécifiques rencontrés

1. Paramètres de recherche

Le problème initial était que lors d'une recherche, lorsqu'il y avait plus d'une page de résultats et qu'on passait à la page suivante, les paramètres de recherches étaient perdus. La première solution fut de passer les paramètres de recherches dans les URLS. On pouvait alors se retrouver dans des cas avec des URLs exotiques, des paramètres entre des slashes, du genre Classification/TypeC/AfficherTessonsClasses. Cette solution n'étant bien sûr clairement inadapté, quand d'autres pages arrivèrent avec encore de la pagination et des paramètres de recherche à gérer en même temps, toute cette partie de l'application fut entièrement refaite. Les paramètres sont désormais passés via le protocole GET (en clair dans l'URL) , grâce au paramètre "paramRoute" de l'objet Paginator. On se retrouve donc maintenant avec des adresses beaucoup plus propre du style classification?typeClassifChoisi=Type+C&tessonsClasses=1 Dès lors, il est possible de récupérer ces paramètres de route dans le controller, à chaque fois qu'il est rappelé en changeant de page.

2. Agrandir les images par simple clic

Afficher les images dans l'application était simple, les redimensionner aussi. En revanche, l'idéal aurait été de pouvoir cliquer dessus pour les afficher en taille réelle, afin de permettre à l'utilisateur de mieux se retrouver au milieu de la multitude de tessons. Pour résoudre ce problème, l'outil FancyBox (basé sur du Jquery) a été utilisé. Une fois la bibliothèque importée et placée dans le bon dossier, quelques lignes simples d'un script permettent d'afficher de façon propre et jolie une image en surimpression de l'écran, sans quitter la page actuelle. Paramétrable, cet outil permet d'afficher l'image et son titre de différentes façons, de modifier le comportement à l'ouverture, …, et s'est trouvé être la réponse idéale à l'agrandissement des images par simple clic.

3. Modification de classification

Un autre problème majeur rencontré dans l'application a été la modification des classes, dans l'onglet classification. Comment permettre à l'utilisateur de modifier directement la classe dans la liste déroulante ? Plusieurs pistes ont d'abord été envisagée, comme faire dix formulaires différents avec un bouton submit au bout de la ligne pour chacun d'eux, faire un formulaire contenant 10 selects différents et avec le bouton submit en bas de page, … Toutes les solutions n'étaient pas satisfaisantes, soit à cause des problèmes que cela engendrait, soit à cause de

l'ergonomie.

La solution trouvée avec Matthieu Exbrayat fut de faire un petit script en JQuery, appelé dès qu'une modification avait lieu sur un des select, et appelant une méthode du controller qui ne faisait rien d'autre que lire les paramètres reçus et changer l'attribut en conséquence dans la base de données.

```
152
153     function modifierClasse(sel){
154         $.post("{{ path('lifo_classif_classification_modifier') }}", {selID:sel.name, classe:sel.value});
155     };
156     </script>
157
```

CONCLUSION

Ce stage a été utile de différentes manières, notamment au niveau de la professionnalisation. Même si j'ai effectué de nombreux jobs d'été au cours de mes études universitaires, force est de constater que le développement d'une application web n'a rien à voir avec de la manutention ou du conditionnement. Le travail est très différent de ce que je connaissais, et effectuer mon stage au sein du LIFO m'a permis, couplé à mon expérience de l'usine, de plutôt bien percevoir ce que sera ma vie future. De plus, l'apprentissage du framework Symfony est un sérieux avantage, car nombre d'applications web aujourd'hui l'utilisent, et avoir des connaissances informatiques sortant du cursus universitaire traditionnel ne peut qu'être bénéfique, dans ma future recherche d'emploi. De plus, l'apprentissage de J2E (qui se rapproche de bien des manières de Symfony) lors de la deuxième année de master en sera d'autant facilité. Être confronté à de nouveaux problèmes a été une expérience très enrichissante.

ANNEXES

A. Fonctionnement de Symfony

Plus qu'un véritable cours sur Symfony, cette section de l'annexe est surtout destinées à rappeler quelques principes de Symfony ainsi que son fonctionnement, afin de mieux comprendre le développement de l'application.

1. Routeur

Le routeur étant écrit en YAML, il faut remplacer les tabulations par quatre espaces pour que ça fonctionne. Il se trouve dans Resources/config/routing.yml ou dans src/Bundle/Resources/config/routing.yml.

Les séparateurs de paramètres dans une route sont "/" ou ".".

L'ordre des arguments des actions n'est pas important, il faut que les paramètres aient le même nom que dans le routeur pour qu'ils soient reconnus.

Il existe différents paramètres systèmes :

- ✗ _format (format de la page – html, xml, …)
- ✗ _locale (langue d'affichage de la page)
- ✗ _controller

defaults correspond aux valeurs par défaut des paramètres, qu'ils soient présents ou non dans le path.

La partie *Requirements:* doit être suivie d'une expression régulière.

2. Controller

Le controller renvoie un objet Response au noyau, et Twig met en forme les données que le controller lui donne.

Le controller dans son expression la plus basique est de la forme :

methodeAction(){

```
    $content = $this
        →get('templating')
        →render('bundle:nom:methode.html.twig') ;
    return new Response($content) ;
}
```

Le controller a la possibilité de récupérer les paramètres hors route :

```
public function viewAction(Request $request){
    $tag = $request→query→get('tag') ;
}
```

Si l'on intègre l'objet Request aux paramètres d'une méthode d'un controller, ne pas oublier le use \Request.

Les variables d'URL($_GET) sont récupérées via *$request→query*, tandis que les variables de formulaires($_POST) sont récupérées via *$request→request*.

`if($request→isMethod('Post'))` => Un formulaire a été envoyé

`if($request→isXmlHttpRequest())` => C'est une requête AJAX

`$this→render(vue, array()) ;` => Retourne une réponse avec la vue correspondant et les paramètres

`return $this→redirectToRoute('route')` => Cette méthode prend en paramètres la route et non l'URL.

```
$session = $request→getSession() ;
$session→getFlashBag()→add('name', 'String') ;
```

=> Affiche un message flash dans la session. Un changement de page ou une actualisation entraîne la disparition du message.

3. Vue

Les templates fonctionnent avec un système de triple héritage :

- ✗ layout général (→structure du site, header, footer, …).
- ✗ layout de bundle (parties communes aux pages d'un même bundle).
- ✗ template de page (→contenu central de la page).

Le layout général se trouve dans *app/Resources/views/layout.html.twig*, et pour l'appeler dans les autres pages il faut utiliser *::layout.html.twig*.

On peut également faire de l'inclusion de controller pour avoir un bloc personnalisé :

{{ render(controller("Bundle:Controller:methode")) }}

Si on veut appeler le layout du bundle, il suffit de faire de la même manière que pour le layout général mais préfixé par le nom du bundle.

4. Services

Un service est un objet PHP qui remplit une fonction, associé à une configuration. Cet objet est accessible depuis n'importe où dans notre code.

On peut stocker des paramètres en plus des services dans le conteneur :

paramaters :

    nom_parametre: ma_valeur

services :

On y accède vie $container→getParameter('nomParametre') ;

Un services peut avoir différents arguments :

- ✗ des valeurs normales en YAML (booléens, caractères, …)
- ✗ des paramètres (définis dans parameters.yml) sous la forme %nom_du_parametre%
- ✗ des services (sous la forme "@nom_du_service")

L'ordre des arguments du constructeur est le même que celui de la configuration du service.

5. Entités et Doctrine

Pour générer une entité via la console : php bin/console generate:doctrine:entity

Il faut avoir avant une base de données déjà créée, soit avec PHPMyAdmin, soit via la console avec doctrine:database:create.

On peut rajouter des méthodes, en plus des getters et setters, qui contiennent de la logique métier.

Pour mettre un attribut par défaut dans une entity, on peut passer soit par le constructeur, soit dès la déclaration de l'attribut.

Le mapping, via les annotations, permet d'enregistrer les entités en base de données.

Pour générer les tables : php bin/console doctrine:schema:update

suivi soit de --dump-sql pour voir la requêtes, soit de --force pour exécuter la requête.

Pour modifier une entité, on rajoute dans le fichier la variable avec ses annotations. Pour générer automatiquement les getters et setters via la console : php bin/console doctrine:generate:entities nomBundle:NomEntity

Le service Doctrine est accessible depuis le controller avec $doctrine = $this→getDoctrine() ;

Tout comme le service EntityManager : $em = $this→getDoctrine()→getManager() ;

Pour accéder aux objets depuis la base de données, on utilise le repository : $fooRepository = $em→getRepository(NomBundle:nomEntity') ;

Utilisation standard d'une entité depuis le controller :

**Création de l'entité

$tesson = new Tesson() ;

$tesson→set…

**On récupère l'EntityManager

$em = $this→getDoctrine()→getManager() ;

**On persiste l'entité

$em→persist($tesson) ;

**On flush ce qui a été persisté avant

$em→flush() ;

Il est inutile de faire un persist() si l'entité a été récupérée via Doctrine (avec un find() par exemple), il sert uniquement) donner la responsabilité de l'objet à Doctrine.

- ✗ clear() annule tous les persist()
- ✗ detach($entity) annule le persist() sur l'entité en argumentscontains($entity) true si l'entité est gérée par l'EntityManager
- ✗ refresh($entity) met à jour l'entité dans l'état où elle est dans la base de données
- ✗ remove(entity) supprime l'entité de la base de données au prochain flush().

L'EntityManager sert à manipuler les entités, tandis que les repositories servent à récupérer les entités.

6. Relations

L'entité propriétaire d'une relation est celle qui contient la référence à l'autre entité, dite inverse.

Pour rendre une association non facultative, il faut rajouter @ORM\JoinColumn(nullable=false)

Relation 1..1 :

Entité propriétaire :

class Advert{

/**

* @ORM\OneTOOne(targetEntity="chemin")

*/

Relation n..1 :

Entité propriétaire :

/**

* @ORM\ManyToOne(targetEntity="chemin")

*/

C'est le côté many d'une relation n..1 qui contient la colonne référence et est donc propriétaire.

Relation n..n :

/**

* @ORM\ManyToMany(targetEntity="chemin")

*/

On choisit le propriétaire le plus logique.

7. Repository

Pour accéder au repository depuis un controller :

$repository=$this

→getDoctrine()

→getManager()

→getRepository('nom_bundle:nom_entity') ;

Divers types de reuqêts sont disponibles par défaut :

$entity = $repository→

- ✗ find($id)
- ✗ findAll()
- ✗ findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null). La limite est le

nombre max à prendre, offset définit le début. On peut mettre plusieurs entrées dans le tableau des critères.

- ✗ findOneBy(array $criteria) Pareil que la méthode findBy() mais pour une seule entité.
- ✗ findByX($valeur), remplacer X par le nom d'une propriété de l'entité.
- ✗ findOneByX($valeur)

Il y a possibilité de personnaliser les requêtes faites par le repository d'une entité, grâce au QueryBuilder. Il dispose de plusieurs méthodes pour construire la requête :

```
$qb = $this→createQueryBuilder('t') ;
$qb     →where('t.largeur=:largeur')
        →setParameter('largeur', $largeur)
        →andWhere('t.date<:date')
        →setParameter('date', $date)
        →orderBy('t.date', 'DESC') ;
```

On peut également centraliser des conditions.

Pour la Query, getResult() retourne un tableau d'objets (idéal pour la modification), tandis que getArrayResult() retourne une tableau de tableau (plus pratique pour la lecture). Ca ne change rien pour Twig qui s'adapte. D'autres méthodes existent pour récupérer un scalaire, un seul résultat, …

La commande php bin/console doctrine:query:dql "Requête" permet de tester les requêtes DQL.

B. Nom des colonnes dans l'import CSV

Le nom des colonnes doit être strictement respecté pour que l'import CSV se déroule. Si un nom diffère de la liste ci-dessous, le parsing des colonnes ne marchera pas. L'ordre par contre n'a aucune importance. Les colonnes non citées ne sont pas utilisées.

- ✗ année
- ✗ commentaire
- ✗ dévellopé (la faute d'orthographe a malheureusement été gardé)
- ✗ égal type
- ✗ équi type
- ✗ Fait
- ✗ INDIVIDU
- ✗ largeur
- ✗ longueur
- ✗ LOT
- ✗ molette référence
- ✗ N° enregistrement
- ✗ N° isolation 1
- ✗ N° PERIODE
- ✗ N° PHASE
- ✗ N° SEQUENCE
- ✗ nbre motifs v
- ✗ nbre motifs h
- ✗ nom du décor
- ✗ position décor
- ✗ site
- ✗ Tombe
- ✗ TYPE DE DÉCOR
- ✗ US
- ✗ Zone

C. Résumé du premier entretien

Dans cette partie se trouve le résumé du premier entretien avec Sébastien Jesset, introduisant une sorte de premier cahier des charges de l'application. Il peut être important pour comprendre le rôle d'un attribut ou d'une relation spécifiques, et a servi de base au développement de l'application.

L'identification d'un tesson se faisait avec le numéro d'US et le numéro d'isolation, mais il a fallu rajouter le numéro du site pour avoir une unicité (code INSEE de la commune + numero du site fouillé de la commune).

Le caractère moyen d'un tesson est calculé de la sorte ; (longueur + largeur)/(nombre de caractères verticaux + horizontaux). Il n'est pas obligatoire.

La classification se hiérarchise de la sorte : classes principales → classes secondaires → type A, B, C.

egal type signifie une correspondance avec le répertoire des matrices, tandis qu'*equi* s'en approche.

Le fait est le numéro de la structure où le tesson a été trouvé, ensemble chrono-stratigraphique.

Individu ou lots peuvent prendre les valeurs suivantes : incomplet, complet, restitué.

La largeur est la hauteur du tesson, tandis que sa longueur est le développé, complétées avec des inégalités (<, ≥, …) dans le cas où c'est incomplet.

Si la molette n'est pas présente ailleurs : molette référence. Une molette référence a forcément un *egal type* avec le tesson correspondant.

Diagramme stratigraphique :

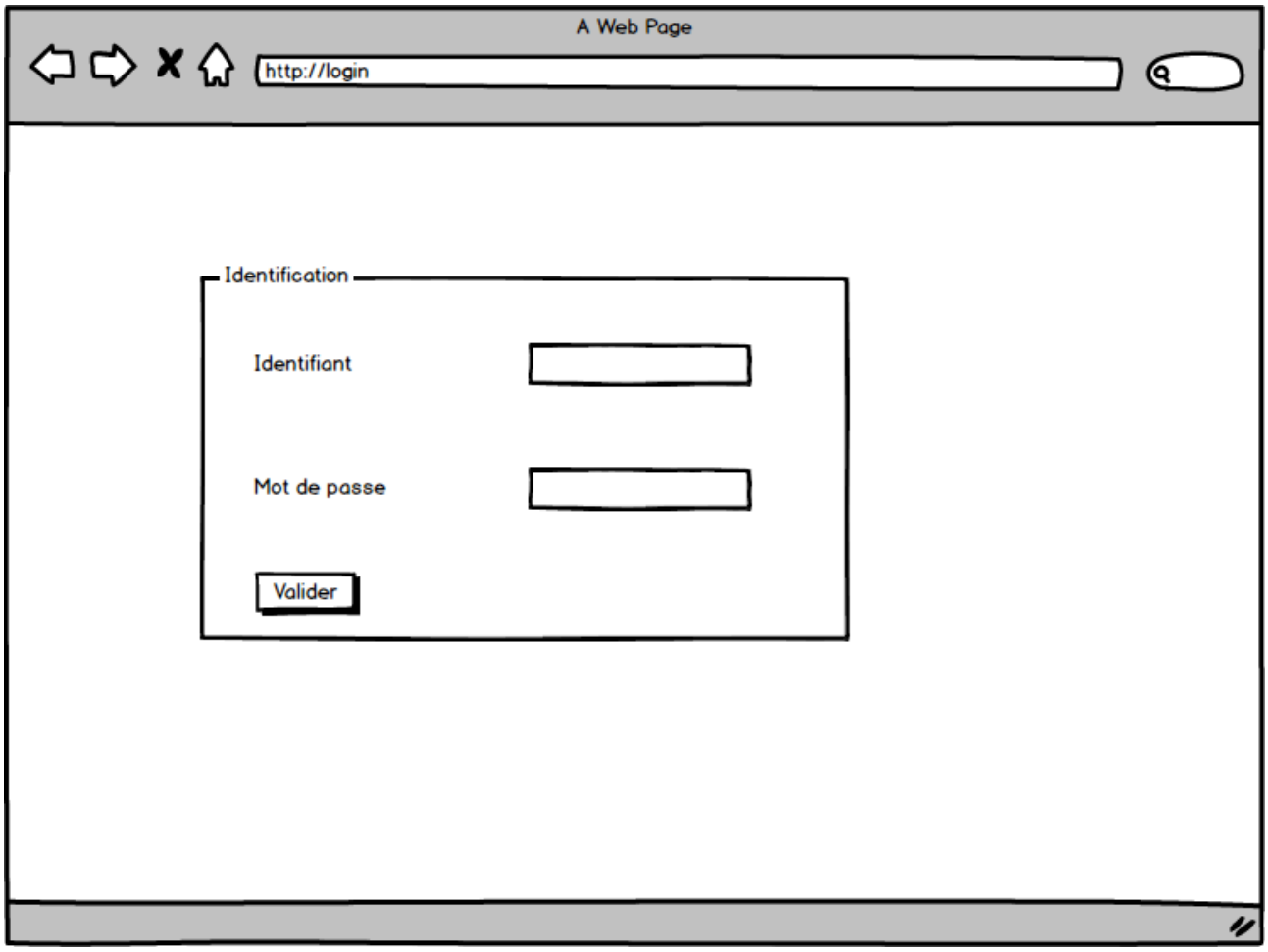
- ✗ séquence : un geste (creuser, combler, …)
- ✗ regroupées en phase : construction, …
- ✗ regroupées en période : bas empire, …

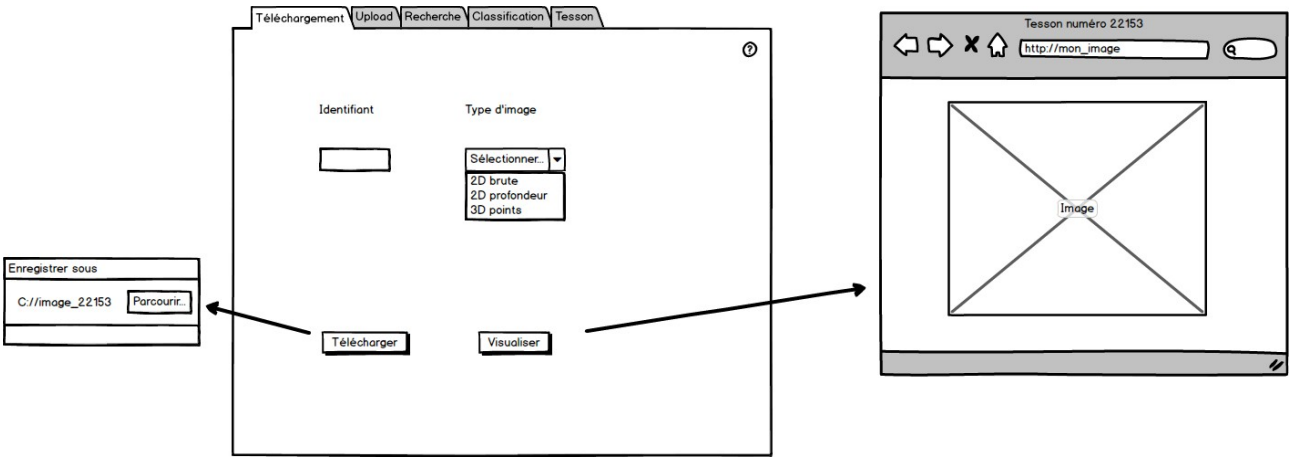
Une zone correspond à un découpage de site.

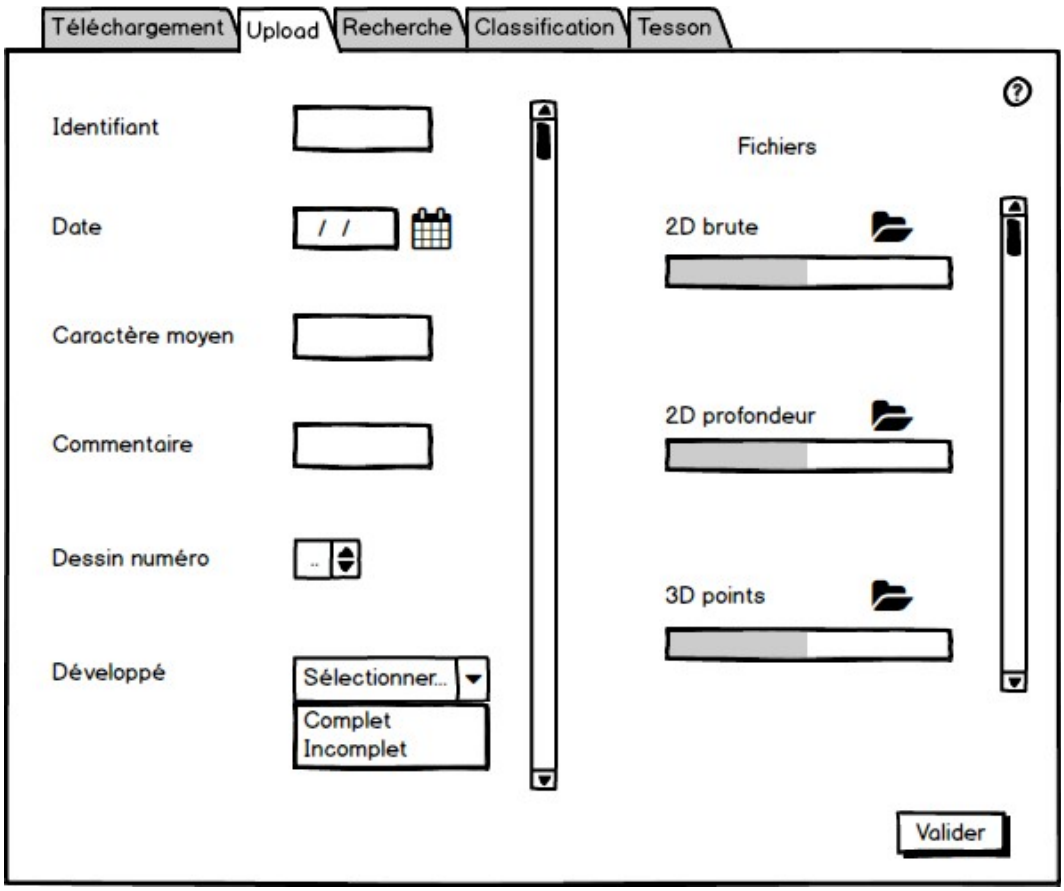
La période ainsi que les autres éléments chronologique sont propres au site.

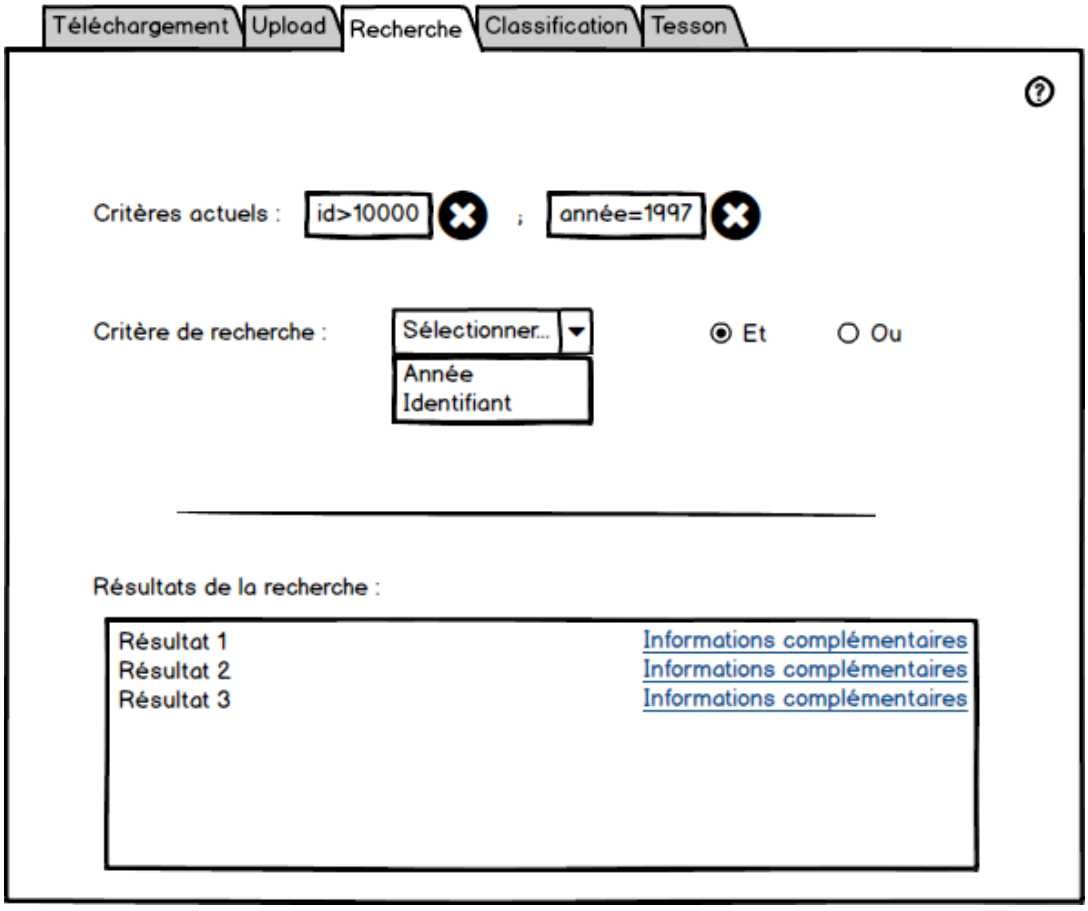
D. Wireframe

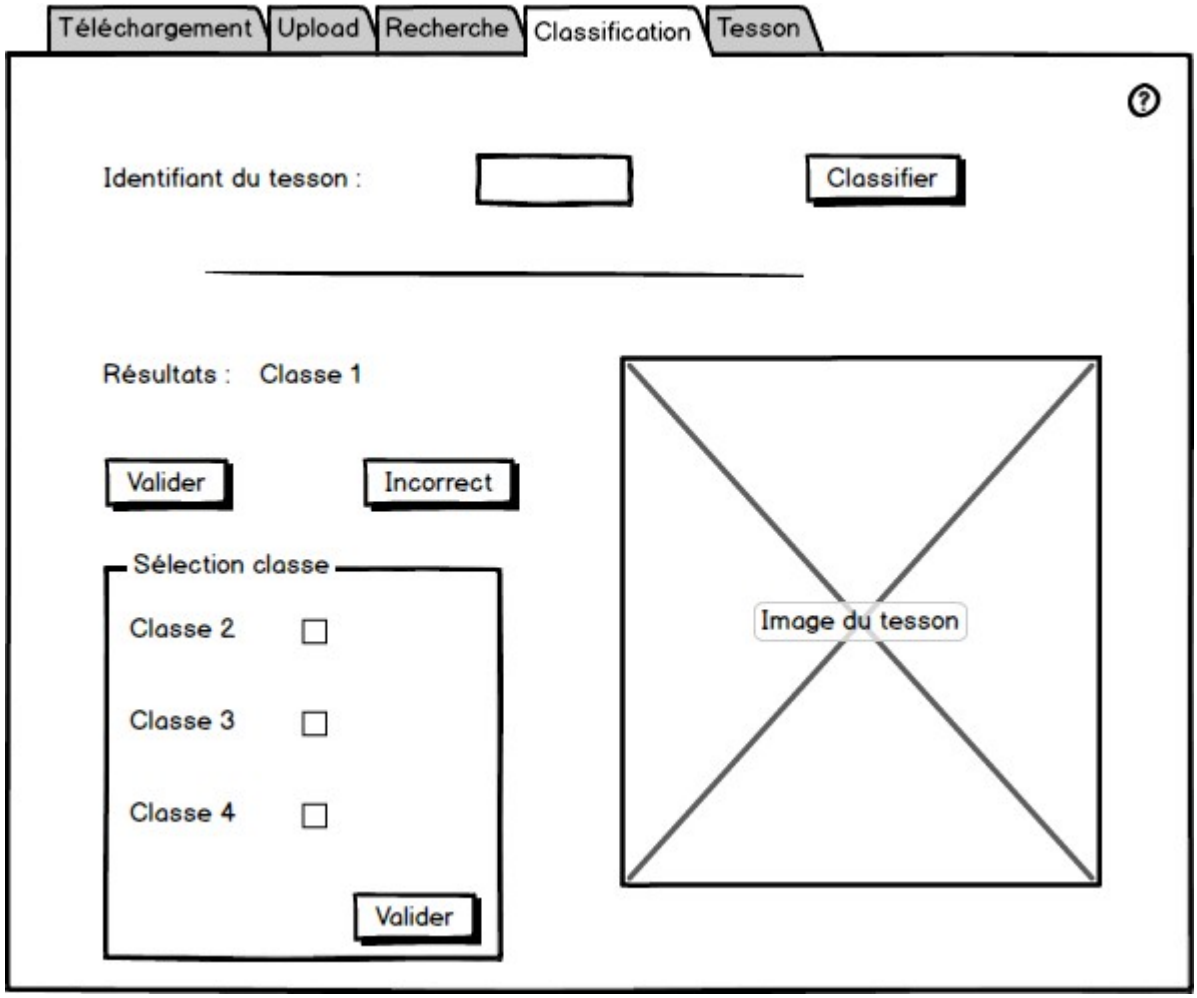
Voici la liste complète des écrans réalisé dans le cadre du wireframe au début du projet.

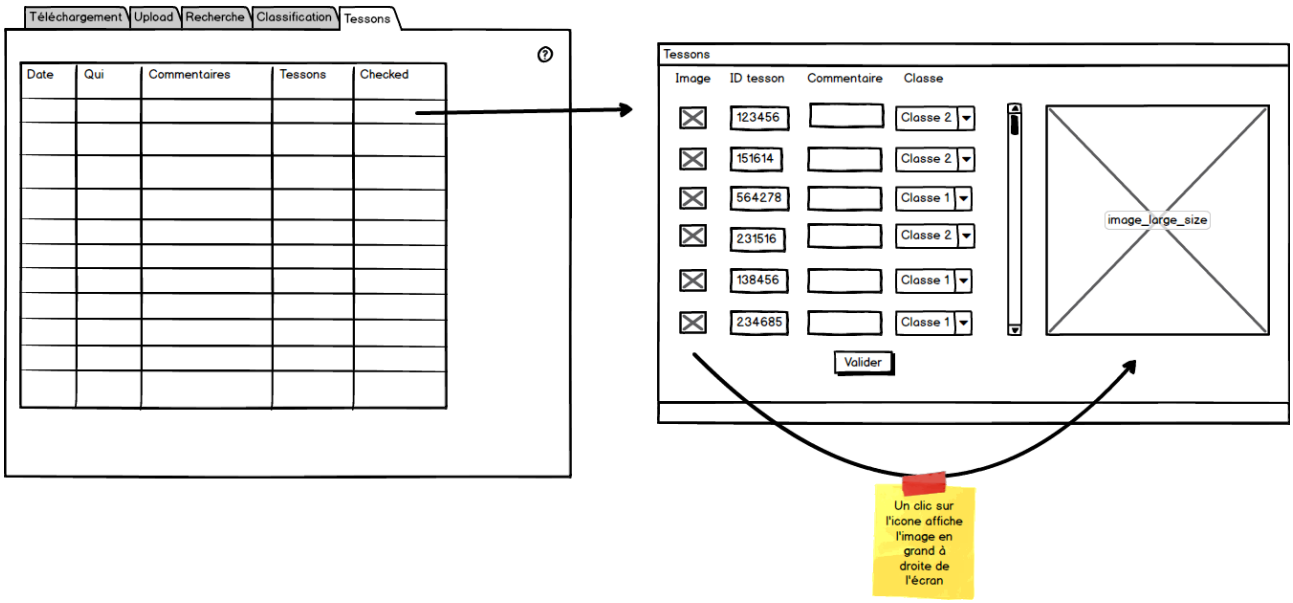





Téléchargement
Upload
Recherche
Classification
Tesson
Identifiant
Date
/ /
Caractère moyen
Commentaire
Dessin numéro
Développé
Sélectionner...
Complet
Incomplet
Fichiers
2D brute
2D profondeur
3D points
Valider


Téléchargement
Upload
Recherche
Classification
Tesson
Critères actuels :
id>10000
;
année=1997
Critère de recherche :
Sélectionner...
Année
Identifiant
Et
Ou
Résultats de la recherche :
Résultat 1
Résultat 2
Résultat 3
Informations complémentaires
Informations complémentaires
Informations complémentaires


Téléchargement
Upload
Recherche
Classification
Tesson
Identifiant du tesson :
Classifier
Résultats : Classe 1
Valider
Incorrect
Sélection classe
Classe 2
Classe 3
Classe 4
Valider
Image du tesson


Téléchargement
Upload
Recherche
Classification
Tessons
Date
Qui
Commentaires
Tessons
Checked
Tessons
Image
ID tesson
Commentaire
Classe
123456
Classe 2
151614
Classe 2
564278
Classe 1
231516
Classe 2
138456
Classe 1
234685
Classe 1
Valider
image_large_size
Un clic sur l'icone affiche l'image en grand à droite de l'écran