

TITRE PROFESSIONNEL

Concepteur, Développeur d'Applications

SOMMAIRE

I/ Liste des compétences du référentiel.....	4
II/ Expression des besoins du projet.....	5
1 - Description du projet.....	5
2 - Utilisateurs du projet.....	5
III/ Développer une application sécurisé.....	5
1 - Installation logiciel et documents nécessaires.....	5
1/ Langages et technologies utilisés.....	6
2/ Environnement de développement avec Docker.....	6
4/ Gestion de version avec Git & GitHub.....	6
3/ IDE et outils de développement.....	8
5/ Base de données et modélisation.....	8
6/ Documentation et suivi avec Notion.....	8
7/ Gestion de projet avec Trello.....	9
2 - Mise en place Frontend Twig et Vue JS.....	9
1/ Intégration de Twig : structure des templates.....	9
2/ Initialisation de Vue.js.....	10
3/ Composant Vue "Badge.vue".....	11
1. Données réactives.....	11
4. Interface utilisateur.....	11
5/ Interaction entre Twig et Vue.js.....	11
3 - Fonctionnalités Inventaire/Badges/Imprimantes.....	12
1/ Inventaire.....	12
Vue côté utilisateur (Frontend Vue.js).....	12
Traitement des données (Backend Symfony).....	13
Importation CSV.....	13
Exportation CSV.....	15
API REST.....	16
Conclusion de l'étape.....	17
2/ Badges.....	17
Interface de gestion des badges (backend Symfony).....	17
Affichage dynamique et gestion côté utilisateur.....	19
Consultation publique côté élèves.....	20
Structure technique et choix d'implémentation.....	20
Conclusion.....	20
3/ Statistiques d'imprimantes.....	21
Interface de gestion des statistiques (Vue.js + API Symfony).....	21
Affichage dynamique des données.....	22
Suivi des actions et messages de débogage.....	23
Structure technique et choix d'implémentation.....	23
Conclusion.....	23
4 - Trello et Github.....	24

1/ Trello.....	24
Vue d'ensemble du tableau Trello.....	25
Exemple de carte Trello.....	25
Suivi des échéances.....	26
2/ GitHub.....	27
IV/ Concevoir et Développer une application sécurisée organisé en couches.....	27
1 - Figma et maquette Edudata.....	27
1/ Les pages conçues (Voir Annexe).....	28
2/ Les composants réutilisables.....	31
3/ L'utilité de cette maquette pour le développement.....	32
2 - Base de donnée logiciel dbdiagram.io / looping.....	32
Objectif de la modélisation.....	32
Utilisation de dbdiagram.io.....	33
Conception avec Looping.....	34
Intégration dans Symfony avec Doctrine.....	35
Conclusion.....	36
3 - EasyAdmin.....	36
1. Redirection automatique.....	36
3. Structure du menu latéral.....	37
Exemple de configuration CRUD : BadgeCrudController.....	38
Avantages de l'approche EasyAdmin.....	38
Conclusion.....	38
VI/ Préparer et déployer une application sécurisée.....	39
1 - Codeception.....	39
2 - Déploiement.....	40
ANNEXES.....	41

I/ Liste des compétences du référentiel

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Développer une application sécurisée	1	Installer et configurer son environnement de travail en fonction du projet
		2	Développer des interfaces utilisateur
		3	Développer des composants métier
		4	Contribuer à la gestion d'un projet informatique
2	Concevoir et développer une application sécurisée organisée en couches	5	Analyser les besoins et maquetter une application
		6	Définir l'architecture logicielle d'une application
		7	Concevoir et mettre en place une base de données relationnelle
		8	Développer des composants d'accès aux données SQL et NoSQL
3	Préparer le déploiement d'une application sécurisée	9	Préparer et exécuter les plans de tests d'une application
		10	Préparer et documenter le déploiement d'une application
		11	Contribuer à la mise en production dans une démarche DevOps

II/ Expression des besoins du projet

1 - Description du projet

EduData est une application web développée avec Symfony pour le backend et Vue.js pour le frontend. Elle a pour objectif de simplifier et d'optimiser la gestion quotidienne d'un établissement scolaire à travers plusieurs fonctionnalités clés, intégrées dans un environnement numérique centralisé et intuitif.

L'une des fonctionnalités principales repose sur l'utilisation de badges. Chaque membre du personnel ou élève peut déclencher un appel visuel via un écran installé dans les locaux de l'établissement. Cette interface permet de cibler efficacement le public concerné, facilitant ainsi la communication interne et l'organisation des flux de personnes.

EduData intègre également un module d'inventaire complet. Celui-ci permet de créer des étiquettes personnalisées pour chaque bien matériel de l'établissement (tables, chaises, ordinateurs, bureaux, etc.). Grâce à ce système, tous les actifs sont répertoriés de manière précise, ce qui facilite leur localisation, leur gestion et leur suivi dans le temps.

Enfin, l'application propose une fonctionnalité dédiée au suivi des imprimantes. Elle permet de monitorer l'ensemble des impressions et numérisations effectuées par le personnel. Cette traçabilité offre une meilleure maîtrise des consommables et renforce la transparence quant à l'usage des ressources partagées.

Avec EduData, l'établissement scolaire bénéficie d'une solution moderne, efficace et évolutive, qui contribue à une meilleure organisation interne, à une réduction des pertes matérielles et à un pilotage plus fin des équipements. L'application répond ainsi aux besoins actuels de digitalisation et d'automatisation des processus dans le secteur éducatif.

2 - Utilisateurs du projet

L'application EduData est utilisée par les services de l'administration, de la comptabilité et du personnel au sein de l'établissement scolaire. Elle permet à chacun de ces rôles de gérer les informations liées à leur domaine

III/ Développer une application sécurisé

1 - Installation logiciel et documents nécessaires

Dans le cadre de mon projet de bachelor en développement, j'ai mis en œuvre un environnement de travail technique complet, pensé pour optimiser la productivité, garantir la

stabilité du projet et suivre une organisation proche des standards professionnels. Voici les différentes composantes que j'ai installées et configurées, ainsi que leur rôle dans le bon déroulement du développement.

1/ Langages et technologies utilisés

Le projet repose sur deux principales technologies :

- **Symfony (PHP)** pour le backend,
- **Vue.js (JavaScript)** pour le frontend.

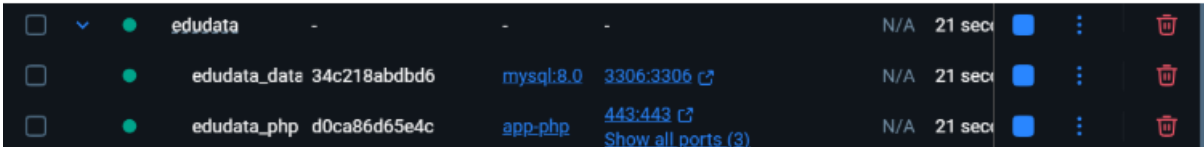
Le choix de **Symfony** s'est justifié par sa robustesse, son écosystème mature, et ses nombreux outils intégrés comme Doctrine, le système de routing ou les commandes CLI. **Vue.js** a été choisi pour sa simplicité de prise en main et sa capacité à créer des interfaces réactives rapidement.

2/ Environnement de développement avec Docker

Afin d'assurer un environnement de développement cohérent et reproductible, j'ai utilisé **Docker** pour conteneuriser l'ensemble du projet. Cela m'a permis d'éviter les problèmes de compatibilité entre les machines, et de garantir que le projet fonctionne de la même manière en local et en production.

J'ai mis en place un fichier `docker-compose.yml` avec plusieurs services :

- Un conteneur **PHP** pour exécuter Symfony,
- Un conteneur **MySQL** pour la base de données,



<input type="checkbox"/>	▼	●	edudata	-	-	-	N/A	21 sec	■	⋮	🗑
<input type="checkbox"/>		●	edudata_data	34c218abdbd6	mysql:8.0	3306:3306 ↗	N/A	21 sec	■	⋮	🗑
<input type="checkbox"/>		●	edudata_php	d0ca86d65e4c	app-php	443:443 ↗ Show all ports (3)	N/A	21 sec	■	⋮	🗑

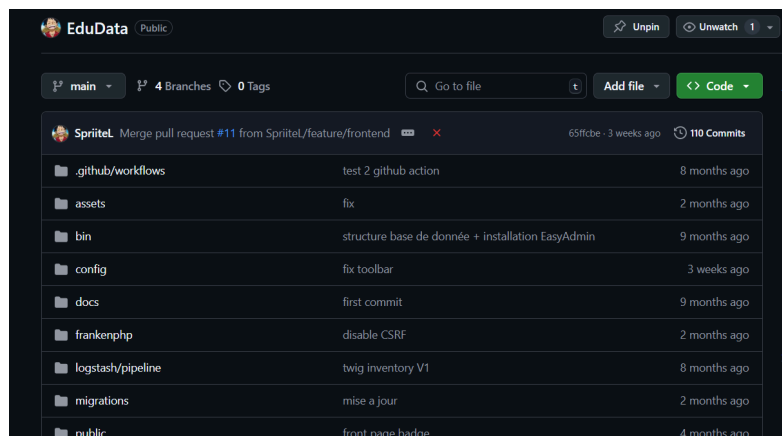
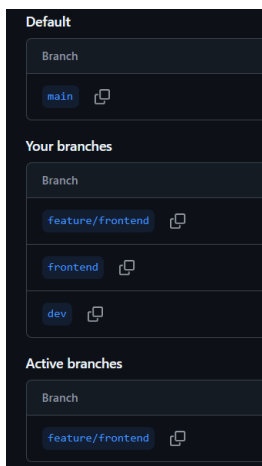
Des volumes ont été configurés pour synchroniser les fichiers du projet avec les conteneurs, ce qui permet de voir immédiatement les changements lors du développement. Des ports ont également été exposés pour accéder à l'application en local via un navigateur (`localhost:8000` par exemple).

J'ai aussi utilisé l'extension **Docker de VS Code**, qui permet de visualiser et gérer facilement les conteneurs directement depuis l'interface de l'IDE (démarrage, arrêt, logs...).

4/ Gestion de version avec Git & GitHub

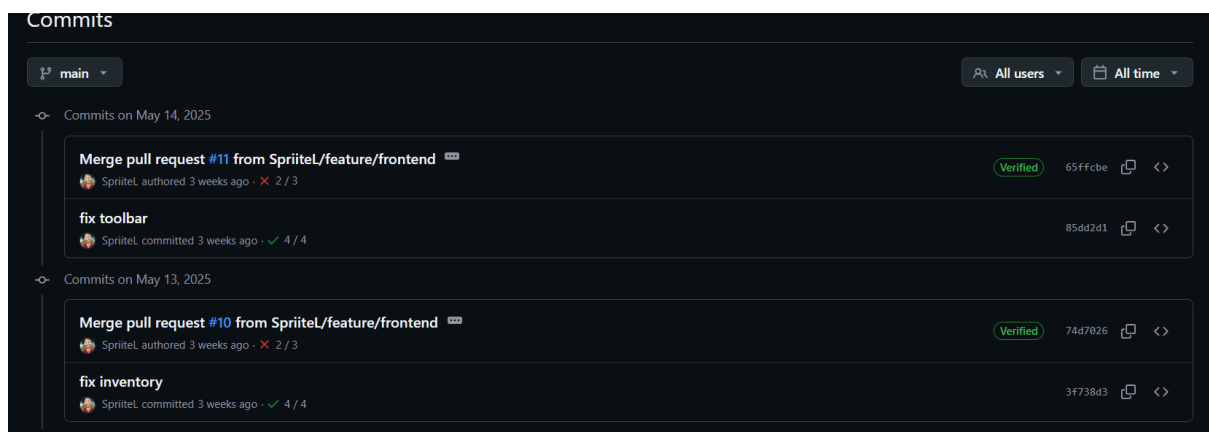
J'ai utilisé **Git** pour le versionning du code, avec **GitHub** comme plateforme d'hébergement. L'organisation du dépôt a été structurée pour permettre un développement clair et collaboratif :

- **main** : la branche stable, contenant le code prêt pour une éventuelle mise en production.
- **feature/fix** : pour le développement de nouvelles fonctionnalités ou de corrections de bugs.
- **frontend** : dédiée au développement de l'interface avec [Vue.js](https://vuejs.org/).
- **dev** : pour d'autres utilisations diverses si nécessaire



Chaque fonctionnalité a été développée dans une branche spécifique issue de **feature/fix**, puis intégrée à **main** via une **pull request**. Ce processus m'a permis de garder un historique clair, de relire chaque portion de code avant intégration, et d'assurer une stabilité constante du projet.

Des **commits réguliers et bien commentés** ont été réalisés tout au long du développement pour documenter l'évolution du projet.



3/ IDE et outils de développement

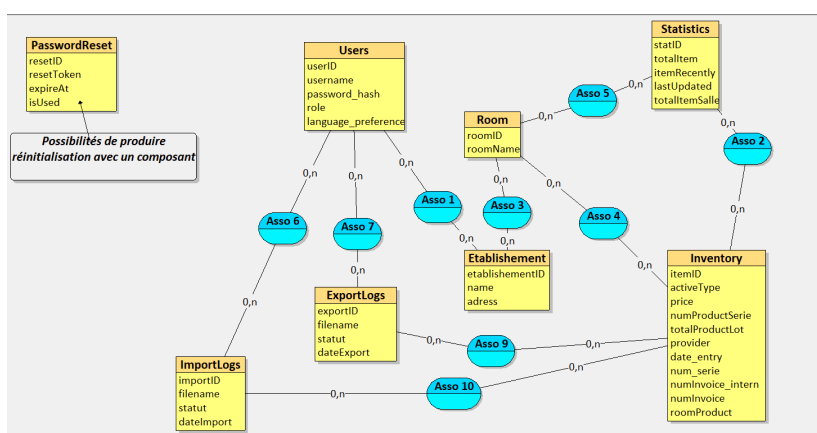
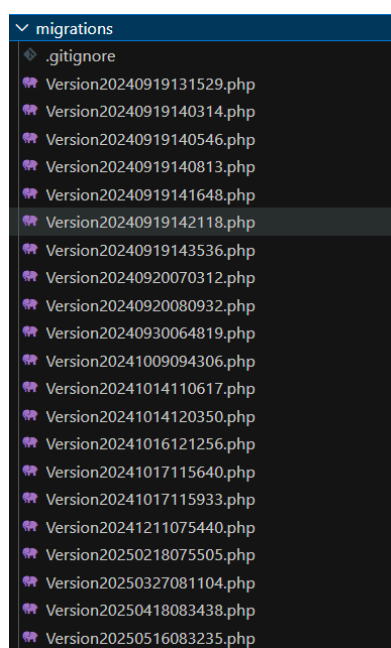
J'ai utilisé Visual Studio Code comme éditeur principal pour son efficacité et sa flexibilité. Mon environnement a été optimisé avec trois extensions clés :

- **Symfony Tools** et **Symfony Container View** pour faciliter le développement avec Symfony (autocomplétion, navigation dans les services, etc.)
- **Docker** pour gérer facilement les conteneurs depuis l'éditeur.

5/ Base de données et modélisation

Le projet utilise une **base de données MySQL**, hébergée dans un conteneur Docker. La structure de la base a été modélisée à l'aide de **Looping**, un outil de modélisation visuelle des bases relationnelles. Cette étape m'a permis de bien penser les relations entre les entités dès le départ.

Une fois le schéma validé, j'ai généré les entités Symfony avec Doctrine, et utilisé les **migrations** pour versionner la structure de la base de données.



(Voir Annexe)

6/ Documentation et suivi avec Notion

La documentation du projet a été centralisée dans un espace **Notion**, où j'ai regroupé :

- La structure des technologies utilisées,

- Les commandes Docker et Symfony importantes,
- La procédure d'installation de l'environnement,
- Des captures d'écran de l'interface et de l'architecture.

Cette documentation a joué un rôle clé pour suivre l'avancement et garder une trace claire de toutes les étapes.

7/ Gestion de projet avec Trello

Pour organiser les tâches et planifier le projet, j'ai utilisé un **tableau Trello** basé sur la méthode Kanban. Le tableau était structuré comme suit :

- BackLog
- A Faire
- En Cours
- En Attente de Test/Validation
- Terminé
- Déploiement

Chaque carte représentait une tâche précise (ex: "Authentification", "Import Données (Format CSV)") avec des priorités et parfois des checklists.

Cela m'a permis de travailler de manière **organisée, itérative et visible**, avec une vue d'ensemble toujours claire sur l'état d'avancement.

2 - Mise en place Frontend Twig et Vue JS

Dans le cadre de mon projet web, j'ai intégré les technologies **Twig** et **Vue.js** afin de construire une interface à la fois performante, claire et interactive. Le but de cette architecture était de combiner les avantages d'un rendu serveur via Symfony et Twig, avec les possibilités dynamiques et réactives qu'offre Vue.js.

Ce document décrit concrètement ce que j'ai mis en œuvre dans mon code pour permettre cette intégration. Il s'agit ici d'un projet basé sur Symfony, dans lequel Twig structure les vues HTML et Vue.js rend certains composants dynamiques grâce à des appels asynchrones.

1/ Intégration de Twig : structure des templates

La partie front-end de l'application repose sur le moteur de template **Twig**, fourni par défaut avec Symfony. Twig me permet d'écrire des fichiers HTML dynamiques, qui peuvent inclure des données envoyées par le contrôleur Symfony, des conditions, des boucles, ou encore des blocs personnalisables.

Dans les fichiers concernés, j'ai défini une structure HTML pour les templates. Ces fichiers contiennent également les **liens vers les fichiers CSS et JavaScript** générés par Webpack Encore, ainsi qu'un point d'ancrage `<div id="app">` dans lequel Vue.js vient se monter :

```
{% block stylesheets %}
<link rel="shortcut icon" type="image/png" href="{{ asset('assets/images/logos/favicon.png') }}" />
<link rel="stylesheet" href="{{ asset('css/styles.min.css') }}" />
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/admin-lte@3.1/dist/css/adminlte.min.css">
<link rel="stylesheet" href="https://cdn.datatables.net/1.13.6/css/jquery.dataTables.min.css">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/lipis/flag-icons@7.2.3/css/flag-icons.min.css">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome@5.15.4/css/all.min.css">
{{ encore_entry_link_tags('app') }}
{% endblock %}
```

Dans les templates spécifiques comme `badge/index.html.twig`, j'ai ensuite inséré mes **composants Vue** à l'intérieur de cette `div` :

```
{% block body %}
<div id="sidebar"></div>
<div id="header"></div>
<div id="badge"></div>
<!-- Body Wrapper -->

{% endblock %}
```

Ce système permet à Twig de générer le HTML de base et de laisser Vue.js gérer le contenu dynamique à l'intérieur du composant.

2/ Initialisation de Vue.js

Vue.js a été initialisé dans le fichier JavaScript principal `app.js`, géré par Webpack Encore. J'ai choisi d'utiliser la version 3 de Vue, avec le système de **composants modulaires**. J'ai commencé par importer Vue et créer une application :

Ici, j'ai enregistré un composant personnalisé appelé `#badge`, défini dans le fichier `Badge.vue`. Ce composant est ensuite utilisé dans mon template Twig, ce qui permet à Vue d'être monté dynamiquement.

3/ Composant Vue "Badge.vue"

Ce composant est le cœur de l'interaction dynamique dans la page des badges. J'y ai défini plusieurs fonctionnalités importantes :

1. Données réactives

Grâce à la fonction `data()`, j'ai défini les variables nécessaires à l'interface, comme les différents événements, le nom, prénom etc.. :

```
<script>
export default {
  data() {
    return {
      badges: []
    };
  },
  mounted() {
    this.loadBadges();
    this.setupEventListeners();
    // Si vous utilisez jQuery et DataTables, décommentez ces lignes
    // $(document).ready(() => {
    //   $('#badge-table').DataTable();
    // });
  },
  methods: {
    setupEventListeners() {
      document.getElementById('add-badge').addEventListener('click', this.addBadge);
    },
    loadBadges() {
      fetch('/badge/list')
        .then(response => response.json())
        .then(data => {
          this.badges = data;
          this.renderBadgeTable();
        })
        .catch(error => console.error('Erreur lors du chargement des badges:', error));
    },
  },
}
```

Ces données sont utilisées pour l'affichage des badges et leur gestion de traitement.

4. Interface utilisateur

Dans la partie template (`<template>`), j'ai créé une interface simple mais fonctionnelle un formulaire regroupant le nom, prénom, des listes déroulantes comprenant l'état de traitement et la classe des étudiants concernés, un tableau DataTable et un bouton "Ajouter". L'utilisateur peut ainsi :

- filtrer les utilisateurs par état de traitement ou par nom
- Ajouter un utilisateur
- Modifier l'état de traitement des badges

5/ Interaction entre Twig et Vue.js

Ce projet repose donc sur une architecture **tripartite** :

1. **Twig** structure les pages côté serveur, avec un rendu initial HTML.
2. **Vue.js** prend en charge l'interactivité en utilisant les composants personnalisés, rendus dans Twig via `<div id="app">`.

Cette séparation permet une **modularité** importante, où chaque couche joue un rôle précis :

- **Twig** : structure des pages et rendu initial
- **Vue** : interaction en temps réel, affichage conditionnel

Le travail que j'ai réalisé dans ce projet montre comment combiner efficacement **Twig et Vue.js** au sein d'un framework Symfony. Twig m'a permis de générer les pages HTML côté serveur, de manière propre, organisée et sécurisée. Vue.js, quant à lui, a permis d'ajouter des fonctionnalités dynamiques à l'interface, sans rechargement de page, notamment pour l'ajout de badges dans le tableau et l'affichage en temps réel dans son tableau d'affichage public.

3 - Fonctionnalités Inventaire/Badges/Imprimantes

1/ Inventaire

Dans le cadre de la gestion du parc matériel de l'établissement, j'ai mis en place un système d'inventaire permettant de répertorier tous les équipements physiques (imprimantes, ordinateurs, etc.) par localisation. L'objectif est double :

1. Faciliter la traçabilité des actifs.
2. Générer des étiquettes RFID à imprimer via un logiciel externe (Zebra Designer).

Cette fonctionnalité a été développée avec un **frontend en Vue.js** pour l'interface utilisateur et un **backend Symfony** pour le traitement des fichiers et la gestion des données.

Vue côté utilisateur (Frontend Vue.js)

La partie visible de l'inventaire est structurée autour d'une interface claire et interactive. L'utilisateur accède à une page contenant un formulaire d'importation, exportation, ainsi qu'un tableau DataTable avec les différentes informations voulues.

Pour chaque import, une table est affichée dynamiquement avec les informations suivantes :

- **Type d'actif**
- **Fournisseur**
- **Date d'arrivée**
- **Numéro de série**
- **Numéro de facture**
- **Numéro de facture interne**
- **Prix neuf**
- **Numéro de produit de la série**
- **Nombre total de produits dans le lot**
- **Nom de la salle**

Un **formulaire d'import CSV** est aussi proposé sur la même page. Il permet à un utilisateur d'importer une liste de matériels via un fichier CSV, qui sera ensuite envoyé à une route Symfony. Le formulaire accepte uniquement les fichiers **.csv**, et un bouton "Importer" déclenche l'envoi via méthode POST.

The screenshot shows a web application interface for inventory management. At the top, there's a search bar and a 'Label' button. Below, the 'Inventaire' section contains a file upload area with 'Choisir un fichier' and 'Aucun fichier choisi' buttons, and 'Importer' and 'Exporter' buttons. Below this is a table with the following data:

Type Actif	Fournisseur	Date d'arrivée	Numéro de Série	Numéro de Facture	Numéro Facture Interne	Prix Neuf	Numéro de produit de la série	Nombre total de produits dans le lot	Nom de la Salle	Actions
Serveur	IBM	2024-05-10	ABC12345	INV-2023-001	INT-001	2300	1	4	Info	
Ordinateur	Dell	2024-05-10	ABC12343	INV-2023-001	INT-001	1200	1	1	Prof	
Test	Beaup	2024-05-10	ABC12341	INV-2023-001	INT-001	342	1	3	Test	

At the bottom of the table area is a 'Recharger' button.

(Voir Annexe)

Importation CSV

Le contrôleur Symfony **InventoryController** contient une méthode dédiée à l'import (**/inventory/import**). Celle-ci récupère le fichier CSV envoyé depuis le frontend et le lit avec la bibliothèque **League\Csv**.

Les étapes sont les suivantes :

1. Le fichier est ouvert et converti en UTF-8 si besoin.

```
#[Route('/inventory/import', name: 'app_inventory_import', methods: ['POST'])]
public function import(Request $request, ManagerRegistry $doctrine): Response
{
    $file = $request->files->get('file');
    if ($file && $file->isValid()) {
        try {
            $content = file_get_contents($file->getPathname());
            if (mb_detect_encoding($content, 'UTF-8', true) === false) {
                $content = mb_convert_encoding($content, 'UTF-8', 'ISO-8859-1');
            }

            $csv = Reader::createFromStream($content);
            $csv->setDelimiter(';');
            $csv->setHeaderOffset(0);
```

2. Les colonnes sont vérifiées pour s'assurer qu'elles correspondent aux champs requis (type d'actif, fournisseur, date d'arrivée, etc.).

```
$inventory = new Inventory();
try {
    // Génération d'un tag RFID unique
    $inventory->setReference($this->generateRFID());
    $inventory->setActiveType($normalizedRow['Type d'Actif']);
    $inventory->setProvider($normalizedRow['Fournisseur']);
    $inventory->setDateEntry(new \DateTime($normalizedRow['Date d'arrivée']));
    $inventory->setNumSerie($normalizedRow['Numéro de Série']);
    $inventory->setNumInvoiceIntern($normalizedRow['Numéro Facture Interne']);
    $inventory->setNumInvoice($normalizedRow['Numéro de Facture']);
    $inventory->setPrice(floatval($normalizedRow['Prix Neuf']));
    $inventory->setNumProductSerie($normalizedRow['Numero de produit de la série']);
    $inventory->setTotalProductLot(intval($normalizedRow['Nombre total de produits dans le lot']));
    $inventory->setNameRoom($normalizedRow['Nom de la Salle']);

    $inventory->setEstablishment($this->getUser()->getEstablishment());

    $entityManager->persist($inventory);
    $count++;
```

3. Chaque ligne du fichier est utilisée pour créer un nouvel objet **Inventory**.
4. Une **référence RFID unique** est générée pour chaque enregistrement via une fonction **generateRFID()** qui crée une chaîne aléatoire de 12 caractères alphanumériques.

```
private function generateRFID(): string
{
    $characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
    $rfid = '';
    for ($i = 0; $i < 12; $i++) {
        $rfid .= $characters[random_int(0, strlen($characters) - 1)];
    }
    return $rfid;
}
```

5. Les données sont ensuite enregistrées dans la base grâce à l'EntityManager.

```
$entityManager->persist($inventory);
$count++;
```

Cette étape permet donc d'alimenter automatiquement la base de données d'inventaire depuis un fichier Excel/CSV issu d'un audit ou d'une collecte sur le terrain.

Exportation CSV

La méthode `export()` du même contrôleur permet de générer un fichier CSV contenant tous les matériels d'un établissement.

Le fonctionnement :

- L'utilisateur authentifié voit uniquement les inventaires liés à son établissement.
- Pour chaque enregistrement, une ligne CSV est générée avec toutes les informations nécessaires à l'impression de l'étiquette RFID.
- Si le produit appartient à un lot, chaque produit de ce lot aura une ligne unique avec une incrémentation du numéro de produit.

Le fichier CSV généré est encodé en UTF-8 (avec BOM) pour être compatible avec **Zebra Designer**, le logiciel utilisé pour l'impression des étiquettes RFID. Une fois généré, il est automatiquement téléchargé.

```
#[Route('/inventory/export', name: 'app_inventory_export', methods: ['GET'])]
public function export(ManagerRegistry $doctrine): Response
{
    $entityManager = $doctrine->getManager();

    // Récupérer l'établissement de l'utilisateur
    $user = $this->getUser();
    $establishment = $user->getEstablishment();
    $inventories = $doctrine->getRepository(Inventory::class)->findBy(['establishment' => $establishment]);

    $csvFileName = 'inventaire_' . date('Ymd') . '.csv';
    $outputBuffer = fopen('php://temp', 'r+');

    fwrite($outputBuffer, "\xEF\xBB\xBF");

    fputcsv($outputBuffer, [
        "Référence",
        "Type d'Actif",
        "Fournisseur",
        "Date d'arrivée",
        "Numéro de Série",
        "Numéro Facture Interne",
        "Numéro de Facture",
        "Prix Neuf",
        "Nombre de Produits",
        "Nombre Total de Produits",
        "Nom de la Salle"
    ], ';');

    foreach ($inventories as $inventory) {
```

API REST

En complément de l'import/export, une API REST a été mise en place :

- `/inventory/api/inventories` : retourne tous les inventaires de l'établissement en JSON.
- `/inventory/api/inventories/{id}` : permet de supprimer un enregistrement spécifique via une requête DELETE.

```
#[Route('/inventory/api/inventories', name: 'api_inventories', methods: ['GET'])]
public function apiInventories(ManagerRegistry $doctrine, SerializerInterface $serializer): Response
{
    $user = $this->getUser();
    $establishment = $user->getEstablishment(); // Récupération de l'établissement de l'utilisateur

    $inventories = $doctrine->getRepository(Inventory::class)->findBy(['establishment' => $establishment]);

    // Sérialisation des données en JSON
    $jsonContent = $serializer->serialize($inventories, 'json');

    return new Response($jsonContent, 200, ['Content-Type' => 'application/json']);
}

#[Route('/inventory/api/inventories/{id}', name: 'api_delete_inventory', methods: ['DELETE'])]
public function apiDeleteInventory(int $id, ManagerRegistry $doctrine): Response
{
    $entityManager = $doctrine->getManager();
    $inventory = $doctrine->getRepository(Inventory::class)->find($id);

    if ($inventory) {
        $entityManager->remove($inventory);
        $entityManager->flush();
        return $this->json(['message' => 'Inventaire supprimé avec succès'], 200);
    }

    return $this->json(['message' => 'Inventaire non trouvé'], 404);
}
```


Ces routes peuvent être utilisées dans des interfaces plus avancées, comme un tableau interactif ou une application mobile, pour gérer l'inventaire en temps réel.

Conclusion de l'étape

La gestion d'inventaire est un module central du projet. Elle permet :

- Un **suivi structuré et automatisé** des biens matériels.
- Une **génération sécurisée et unique de tags RFID**.
- Une **interopérabilité avec des outils externes** comme Zebra Designer pour l'étiquetage.

Cette solution rend les processus d'audit, de maintenance et de gestion du parc matériel plus rapides, fiables et traçables.

2/ Badges

Dans le cadre du projet de gestion numérique des badges au sein de l'établissement, une fonctionnalité de suivi des badges a été développée afin d'améliorer la gestion des demandes et la communication avec les élèves. Cette fonctionnalité vise à centraliser l'enregistrement, le traitement et la consultation des badges à remettre, avec un double objectif :

- Faciliter la gestion des badges par le personnel administratif.
- Offrir aux élèves un accès simple et rapide à l'état d'avancement de leur demande de badge.

Cette fonctionnalité est intégrée au backend Symfony, avec un frontend en JavaScript natif pour les interactions utilisateur.

Interface de gestion des badges (backend Symfony)

La partie gestion, accessible au personnel, permet d'enregistrer et de suivre chaque badge demandé. L'interface repose sur un formulaire simple permettant la saisie des informations suivantes :

- Nom et prénom de l'élève.
- Classe.
- Date prévue pour le retrait du badge.

- Lieu de retrait.
- Statut du badge (par exemple : en cours, traité, annulé).

Le formulaire envoie les données via une requête à une route Symfony dédiée. Le contrôleur reçoit les données et utilise Doctrine ORM pour enregistrer ou modifier les informations dans la base de données.

Le contrôleur principal, `BadgeController`, comporte plusieurs méthodes clés :

- **Ajout et modification** : réception des données via POST, création ou mise à jour d'une entité `Badge`, et persistance en base.

```
#[Route('/badge/add', name: 'badge_add', methods: ['POST'])]
public function addBadge(Request $request, EntityManagerInterface $entityManager): JsonResponse
{
    $data = $request->request;

    // Création du badge
    $badge = new Badge();
    $badge->setNom($data->get('nom'));
    $badge->setPrenom($data->get('prenom'));
    $badge->setDate(new \DateTime($data->get('dateTraitement')));
    $badge->setClasse($data->get('classe'));
    $badge->setEtatTraitement($data->get('etatTraitement'));
    $badge->setLieu($data->get('lieu'));

    // Sauvegarde dans la base
    $entityManager->persist($badge);
    $entityManager->flush();

    // Retourne le badge ajouté
    return new JsonResponse([
        'id' => $badge->getId(),
        'nom' => $badge->getNom(),
        'prenom' => $badge->getPrenom(),
        'dateTraitement' => $badge->getDate()->format('Y-m-d'),
        'classe' => $badge->getClasse(),
        'etatTraitement' => $badge->getEtatTraitement(),
        'lieu' => $badge->getLieu(),
    ]);
}
```

- **Suppression** : suppression d'un badge via son identifiant.

```
#[Route('/badge/delete/{id}', name: 'badge_delete', methods: ['DELETE'])]
public function deleteBadge(int $id, EntityManagerInterface $entityManager): JsonResponse
{
    // Récupérer le badge par son ID
    $badge = $entityManager->getRepository(Badge::class)->find($id);

    if (!$badge) {
        return new JsonResponse(['error' => 'Badge non trouvé'], 404);
    }

    // Supprimer l'entité
    $entityManager->remove($badge);
    $entityManager->flush();

    return new JsonResponse(['success' => true]);
}
```

- **Récupération** : extraction des badges stockés pour un affichage dynamique dans l'interface.

```
#[Route('/badge/treated', name: 'badge_treated', methods: ['GET'])]
public function getTreatedBadges(EntityManagerInterface $entityManager): JsonResponse
{
    $badges = $entityManager->getRepository(Badge::class)->findBy(['etatTraitement' => 'Traitée']);

    $data = [];
    foreach ($badges as $badge) {
        $data[] = [
            'nom' => $badge->getNom(),
            'prenom' => $badge->getPrenom(),
            'classe' => $badge->getClasse(),
        ];
    }

    return new JsonResponse($data);
}

#[Route('/public/badges', name: 'public_badges', methods: ['GET'])]
public function publicDisplay(): Response
{
    return $this->render('badge/public_badges.html.twig');
}
```

Ces opérations sont réalisées en JSON, ce qui permet une mise à jour en temps réel de la liste sans rechargement complet de la page.

Affichage dynamique et gestion côté utilisateur

Un tableau dynamique présente en temps réel les badges en attente ou traités. Grâce à l'utilisation de la fonction JavaScript `fetch()`, les données sont récupérées depuis l'API Symfony au format JSON et affichées via manipulation du DOM. Cela permet au personnel d'avoir une vue à jour des badges sans recharger la page.

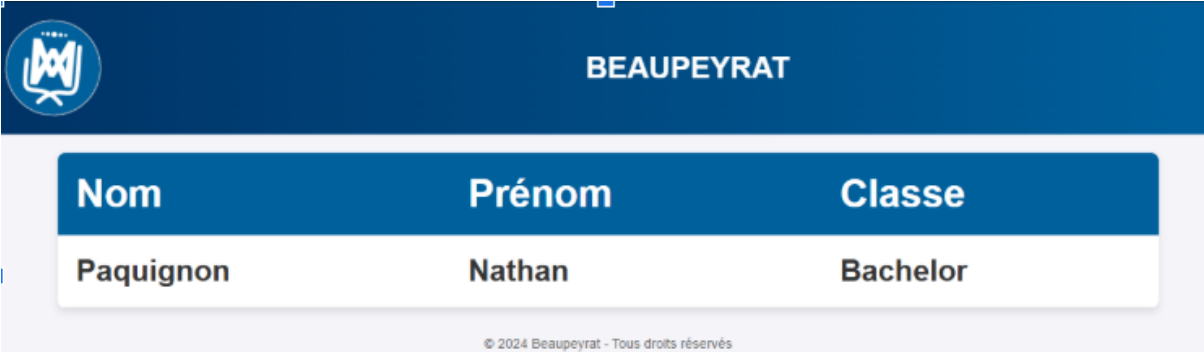
Toutes les 10 secondes une actualisation automatique du tableau est effectuée. Cette interaction fluide améliore l'expérience utilisateur et réduit les erreurs dues aux données obsolètes.

Consultation publique côté élèves

Une interface publique affichée dans les couloirs de l'établissement a été mise en place pour que les élèves puissent consulter l'état de leur demande de badge. Cette page affiche uniquement les badges dont le statut est "traité", indiquant que le badge est prêt à être retiré.

La page interroge régulièrement, toutes les 10 secondes l'actualisation des badges, afin de fournir une liste toujours actualisée. L'affichage reprend les informations suivantes :

- Nom et prénom de l'élève.
- Classe.



Nom	Prénom	Classe
Paquignon	Nathan	Bachelor

© 2024 Beaupeyrat - Tous droits réservés

Structure technique et choix d'implémentation

- **Backend Symfony** : le framework Symfony assure la gestion des routes, la persistance des données via Doctrine ORM, et la sérialisation des entités en JSON.
- **Base de données** : une entité **Badge** représente chaque demande, avec des champs strictement définis pour garantir l'intégrité des données.
- **Frontend JavaScript/Vue JS** : un interface léger, gère les appels API, la mise à jour du DOM et les interactions utilisateur.
- **Sécurité** : l'interface d'administration est prévue pour être sécurisée via l'authentification. La page publique reste libre d'accès.-

Conclusion

Cette fonctionnalité de suivi des badges constitue un outil essentiel pour la gestion administrative des demandes de badges dans l'établissement. Elle permet :

- Un enregistrement structuré et centralisé des demandes.
- Une consultation en temps réel par le personnel et les élèves.
- Une réduction significative des interruptions et appels sonores liés aux questions sur l'état et la récupération des badges.

Intégrée de manière fluide dans le projet global, cette fonctionnalité contribue à améliorer l'efficacité et la communication entre élèves et administrations.

3/ Statistiques d'imprimantes

Dans le cadre du projet de gestion numérique des équipements informatiques, une fonctionnalité de suivi et de gestion des statistiques d'imprimantes a été développée. Celle-ci permet de centraliser les données d'impression, de les consulter facilement et de les exploiter à des fins de suivi ou d'optimisation.

L'objectif est double :

- Offrir une interface de visualisation claire et accessible pour le personnel administratif et technique.
- Permettre une gestion simplifiée des statistiques, avec des options d'import/export et de nettoyage des données.

Cette fonctionnalité repose sur un backend Symfony, ainsi qu'un frontend développé en Vue.js pour une expérience utilisateur fluide et réactive.

Interface de gestion des statistiques (Vue.js + API Symfony)

La partie interface est conçue comme un composant autonome Vue.js intégrée au sein d'une page d'administration. elle permet d'effectuer les actions suivantes :

- Import de fichiers CSV contenant les statistiques.
- Export des données au format CSV pour archivage ou analyse.
- Affichage des statistiques existantes sous forme de tableau.
- Suppression unitaire d'une statistique.
- Affichage détaillé des données brutes d'une ligne de statistique (en JSON).

- Possibilité d'alternier entre une **vue simplifiée** et une **vue avancée** (via DataTables).

Statistiques d'imprimantes

Search

Impressions

Info de débogage: 10:27:47: Stats mises à jour: 0 entrées

Choisir un fichier: Aucun fichier choisi

Importer Exporter Voir données brutes

Passer à la vue avancée

Utilisateur	Total Couleur	Total Noir	Total Scan	Actions
-------------	---------------	------------	------------	---------

Recharger Effacer débogage

(Voir Annexe)

Le formulaire d'import autorise uniquement les fichiers **.csv**. Une fois sélectionné, le fichier est envoyé au backend via une requête **fetch()**. L'API Symfony traite ensuite le contenu, l'encode et l'insère en base via Doctrine.

```
methods: {
  fetchImprimantes() {
    axios.get("/api/imprimantes").then((response) => {
      this.imprimantes = response.data;
    });
  },
  filteredImprimantes(location) {
    return this.imprimantes.filter((item) => item.printer === location);
  },
  mounted() {
    this.fetchImprimantes();
  },
};
/script>
```

Une fois l'import terminé, une notification est affichée et les données sont automatiquement rechargées dans l'interface. Le tableau est alors mis à jour sans rechargement de la page.

Affichage dynamique des données

Deux modes d'affichage sont proposés à l'utilisateur :

- **Vue simplifiée** : affichage d'un tableau HTML classique sans fonctionnalités supplémentaires.
- **Vue avancée (DataTables)** : affichage enrichi avec tri, recherche instantanée et pagination.

Ce basculement est contrôlé par un simple bouton radio qui active ou désactive DataTables via des appels jQuery. Cette flexibilité permet d'adapter l'interface à tous les profils d'utilisateurs, qu'ils soient technique ou non.

Chaque ligne du tableau comprend un bouton "Données brutes" permettant d'ouvrir une modale affichant le contenu brut du champ `stats`, encodé en JSON.

Un second bouton "Supprimer" est également présent sur chaque ligne. Lors du clic, une confirmation est demandée à l'utilisateur avant d'envoyer une requête de suppression à l'API. En cas de succès, la ligne est supprimée du tableau sans rechargement de la page.

Suivi des actions et messages de débogage

Le composant intègre un panneau de debug en bas de l'interface qui affiche en temps réel les étapes et messages du système, tels que :

- Lancement de l'import.
- Réception du fichier par le backend.
- Résultat de l'insertion en base.
- Suppression réussie ou échouée.
- Erreurs côté client ou serveur.

Ces messages sont horodatés, facilitant ainsi le suivi des actions et le diagnostic d'éventuels problèmes.

Structure technique et choix d'implémentation

Frontend Vue.js : composant unique avec état interne, manipulation du DOM, gestion des formulaires, et intégration conditionnelle de DataTables (via jQuery).

Backend Symfony : traitement des fichiers CSV, validation, sérialisation et gestion de la persistance via Doctrine ORM.

Interopérabilité : l'import/export au format CSV permet une compatibilité avec d'autres outils (Excel, LibreOffice, etc.).

Conclusion

Cette fonctionnalité de gestion des statistiques d'imprimantes offre une interface à la fois technique et intuitive pour le suivi de l'activité d'impression. Elle permet :

- Un contrôle et une traçabilité des impressions.
- Une importation et exportation faciles des données.
- Une vue claire et personnalisable selon le profil utilisateur.
- Une meilleure exploitation des données pour des décisions futures (économies de papier, identification des usages excessifs, etc.).

Elle s'intègre pleinement au système de gestion numérique de l'établissement et renforce l'efficacité des processus internes liés à l'impression.

4 - Trello et Github

Pour mener à bien le développement de cette application, j'ai mis en place une stratégie de gestion de projet structurée, en m'appuyant sur des outils numériques adaptés aux différentes phases du travail. L'objectif était de garantir une bonne organisation, un suivi rigoureux de l'avancement, ainsi qu'une cohérence entre la conception graphique et le développement technique.

Dans cette optique, j'ai utilisé **Trello** pour organiser les tâches à accomplir, planifier les différentes étapes et suivre l'évolution du projet en temps réel. Enfin, j'ai utilisé **GitHub** pour gérer le code source, suivre les versions et assurer la sauvegarde régulière de l'ensemble du projet.

Ces deux outils, complémentaires, m'ont permis de structurer mon travail de manière claire et efficace. Ils ont joué un rôle clé dans la bonne conduite du projet, chacun apportant une réponse adaptée à un besoin spécifique de gestion ou de développement.

1/ Trello

Dans le cadre de la gestion de ce projet, j'ai opté pour **Trello**, une application en ligne spécialisée dans l'organisation et le suivi des tâches. Cet outil collaboratif m'a permis de structurer efficacement les différentes étapes du développement, tout en assurant un pilotage clair et organisé du projet.

Trello fonctionne sur un système de **tableaux**, **listes** et **cartes**, permettant de visualiser l'avancement des tâches de manière intuitive. Chaque carte représente une tâche ou une étape du projet, à laquelle il est possible d'associer des descriptions, des dates d'échéance, des pièces jointes ou encore des checklists. Cela facilite le découpage du projet en sous-tâches claires et hiérarchisées.

Grâce à cet outil, j'ai pu :

- **Planifier les différentes phases** du projet en tenant compte des délais et priorités ;
- **Assurer un suivi régulier** de l'état d'avancement (tâches à faire, en cours, terminées) ;
- **Adapter l'organisation** en fonction des évolutions et contraintes rencontrées.

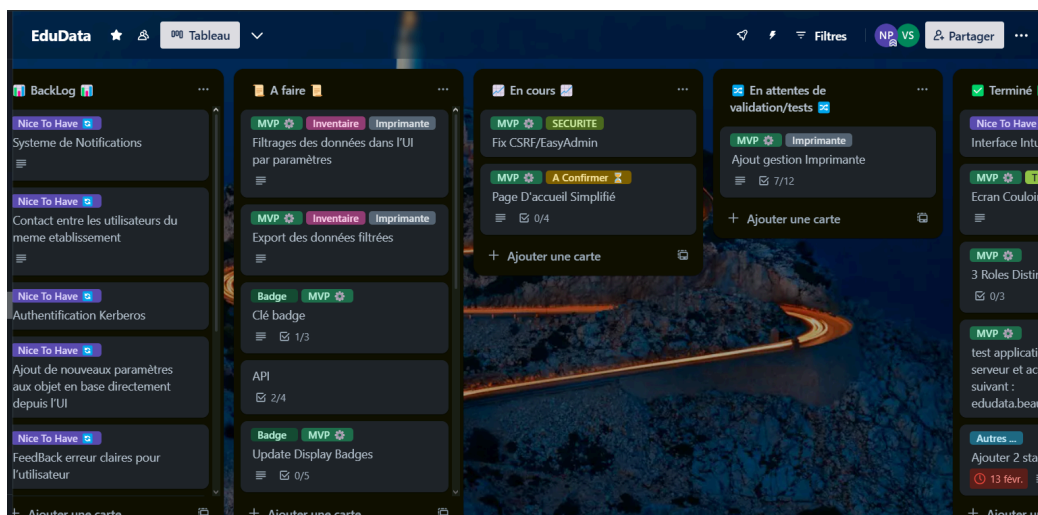
L'utilisation de Trello a contribué à maintenir une vision globale du projet tout en optimisant la gestion du temps et des ressources. Cet outil s'est donc révélé être un véritable atout pour mener à bien le projet dans de bonnes conditions.

Vue d'ensemble du tableau Trello

Le tableau Trello utilisé pour ce projet est organisé en plusieurs listes correspondant aux différentes étapes du processus de développement. On y retrouve notamment les listes :

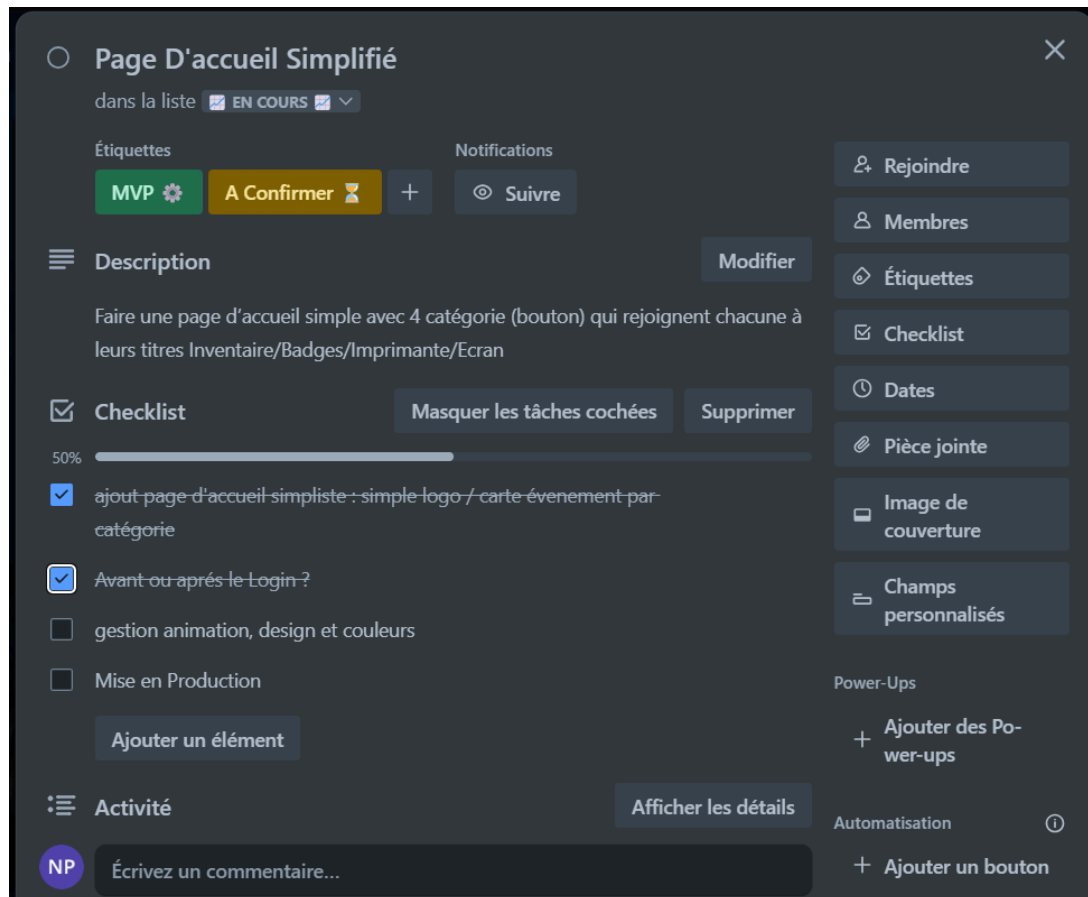
- Backlog : fonctionnalités non-obligatoire mais pouvant apporter de l'ergonomie et de l'UX aux utilisateur
- A Faire : Tâches à réaliser en priorité sur l'application
- En Cours : les tâches actuellement en cours de developpement
- En Attente de Validation/Test : en cours d'essai ou validation du client
- Terminé : tâche terminée mais non déployer
- Déploiement {date}: tâche terminée et déployer à une date, permettant de situer le déploiement concerné

Cette structuration permet de visualiser rapidement l'état global du projet, de hiérarchiser les priorités et de suivre l'évolution des tâches de manière intuitive. Chaque colonne regroupe des cartes représentant des actions précises à accomplir, facilitant ainsi le suivi quotidien.



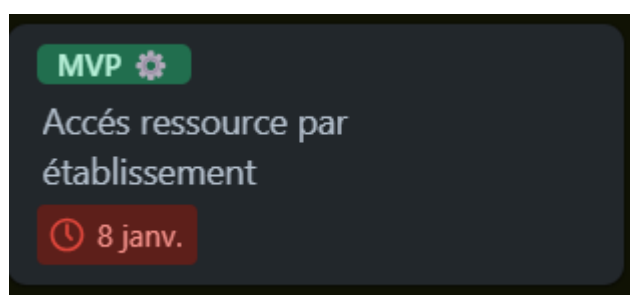
(Voir Annexe)

Chaque tâche est représentée sous forme de carte détaillée, comprenant une description claire des objectifs, une checklist des sous-tâches à réaliser, ainsi que la possibilité d'ajouter des pièces jointes ou des commentaires. Cette organisation permet une gestion granulaire du projet, tout en garantissant une bonne traçabilité des actions à mener. Les cartes sont régulièrement mises à jour pour refléter l'état d'avancement réel.



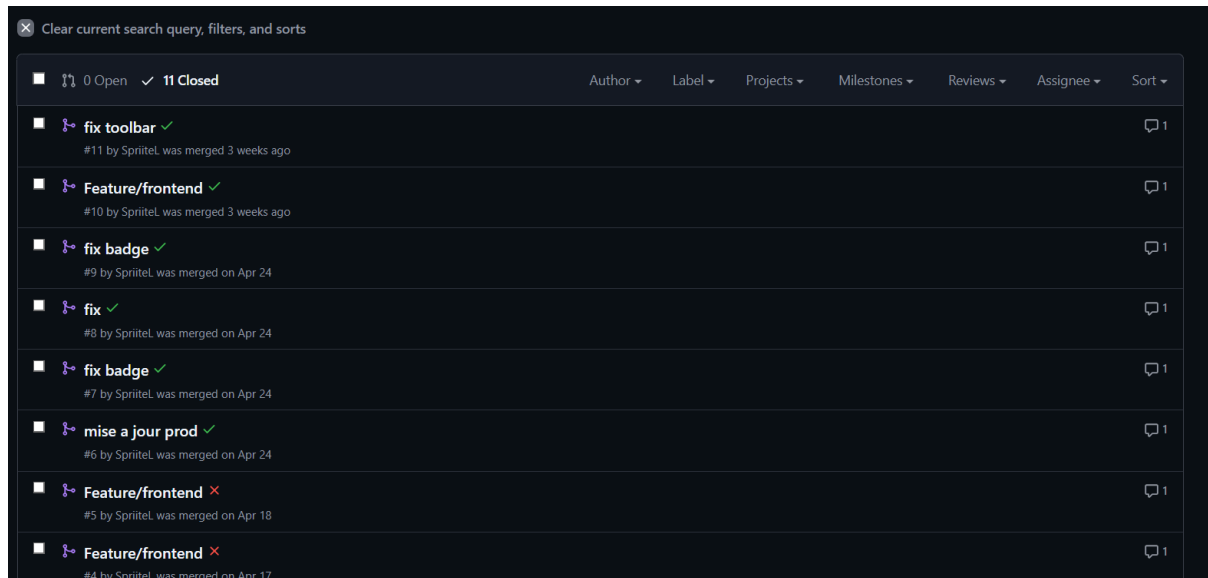
Suivi des échéances

Afin de respecter les délais fixés, chaque carte peut être associée à une date d'échéance. Cette fonctionnalité permet de prioriser les tâches et de maintenir une progression constante dans le temps imparti. Lorsqu'une échéance approche ou est dépassée, Trello signale visuellement le retard, ce qui constitue une aide précieuse pour réajuster le planning si nécessaire. Cela contribue à une gestion du temps plus rigoureuse et efficace.



2/ GitHub

Pour assurer le suivi et la bonne organisation de mon projet, j'ai choisi d'utiliser **GitHub** comme plateforme principale d'hébergement et de gestion du code source. Cet outil m'a permis non seulement de stocker le code de manière sécurisée, mais aussi de suivre l'évolution du projet de façon précise grâce à l'**historique des commits** et aux **pull requests**.



Chaque modification importante est d'abord développée dans une branche secondaire, puis intégrée à la branche **main** via une **pull request**. Ce processus permet de garder un contrôle clair sur les changements apportés, de relire le code avant validation, et de garantir une bonne stabilité du projet.

Grâce à cette organisation, j'ai pu **travailler de manière structurée et méthodique**, tout en gardant une visibilité constante sur l'état d'avancement du projet. L'utilisation des **commits réguliers** m'a également permis de documenter chaque étape du développement, facilitant ainsi la compréhension et le retour en arrière en cas de besoin.

IV/ Concevoir et Développer une application sécurisée organisé en couches

1 - Figma et maquette Edudata

Dans le cadre de mon projet EduData, une des premières étapes a été la conception de l'interface utilisateur via l'outil de maquettage Figma. L'objectif était de réfléchir en amont à l'ergonomie de l'application, à l'organisation des écrans, à la structure de navigation ainsi qu'à l'apparence visuelle des différents éléments d'interface. Cette étape s'est révélée

essentielle pour donner une cohérence d'ensemble à l'expérience utilisateur et guider efficacement le développement frontend avec Vue.js.

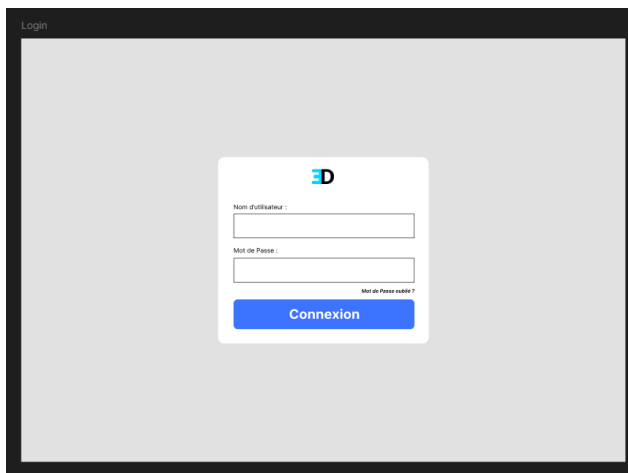
J'ai utilisé Figma non seulement pour dessiner les pages de l'application, mais également pour structurer les composants de base, réfléchir aux parcours utilisateurs et anticiper les contraintes techniques. Cela m'a permis de poser un cadre clair pour tout ce qui concerne l'aspect visuel et interactif de l'application.

La maquette a été pensée dès le départ pour une interface claire, épurée et fonctionnelle, adaptée à une utilisation sur desktop. J'ai donc travaillé en priorité sur les écrans principaux de l'application, tout en créant un système de composants réutilisables (sidebar, header, footer) permettant de maintenir une homogénéité d'un écran à l'autre.

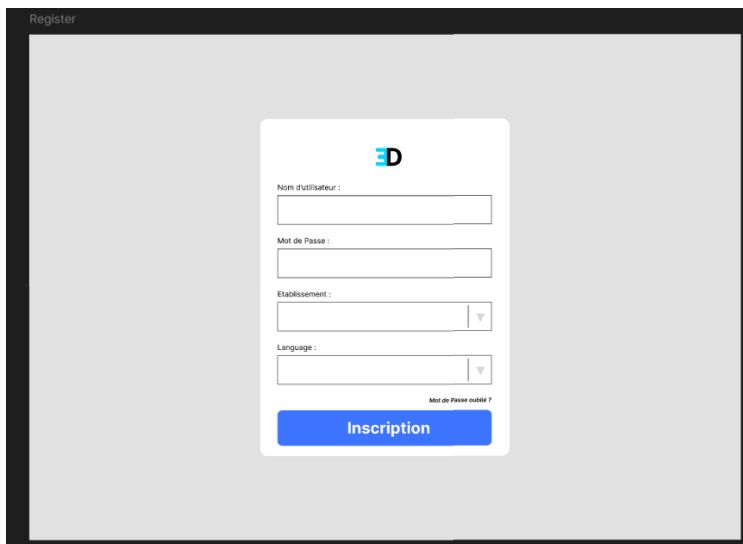
1/ Les pages conçues (Voir Annexe)

J'ai réalisé plusieurs écrans dans Figma, chacun correspondant à une fonctionnalité ou à une section bien précise du projet EduData. Voici un résumé des pages conçues et des choix faits lors de leur élaboration :

- **Page de login** : cette page est volontairement minimaliste, avec un champ pour l'email, un champ pour le mot de passe et un bouton de connexion bien visible. Le design met l'accent sur l'accessibilité avec des champs espacés et un rappel clair en cas d'erreur. L'idée était ici de permettre à l'utilisateur de se connecter rapidement, sans éléments perturbateurs.



- **Page de register** : très proche visuellement de la page de login, elle comprend cependant des champs supplémentaires pour l'adresse mail, le nom, le prénom et l'acceptation des conditions générales d'utilisation. Le formulaire suit une logique verticale simple, avec une hiérarchie visuelle claire entre les titres, les champs et les boutons d'action. J'ai également prévu une indication visuelle pour les erreurs de validation côté formulaire.



Register

ED

Nom d'utilisateur :

Mot de Passe :

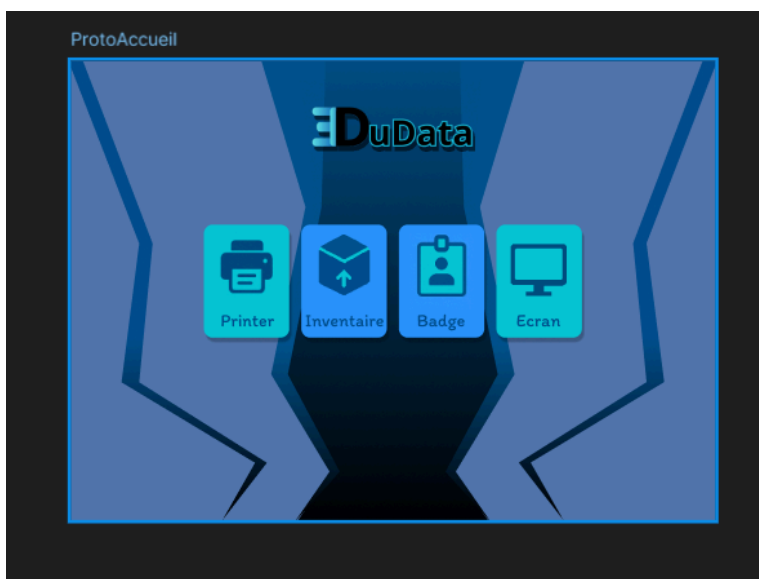
Etablissement :

Language :

Mot de Passe oublié ?

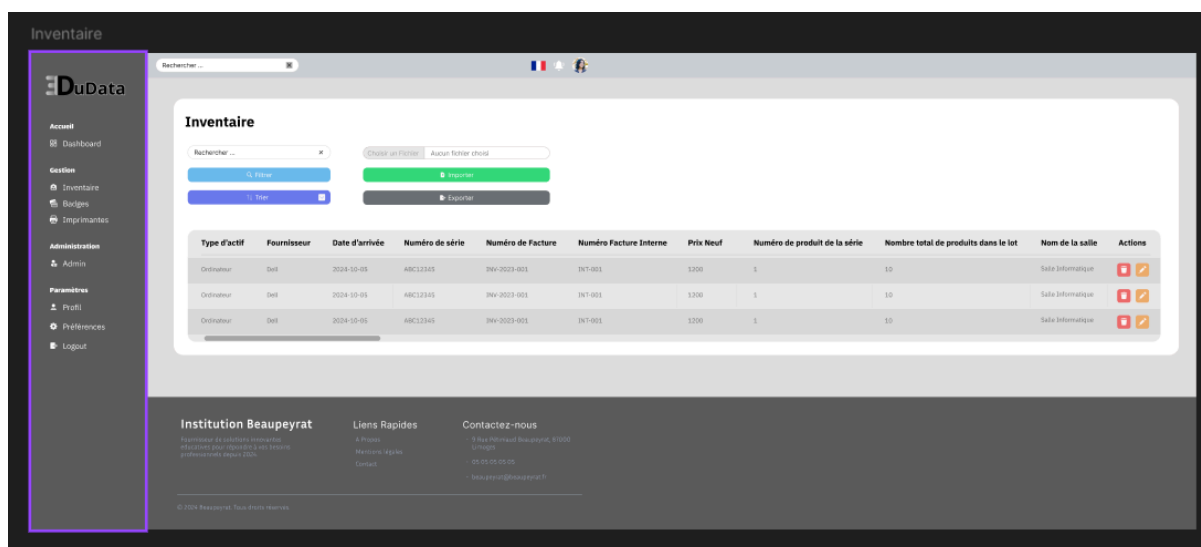
Inscription

- **Page d'accueil** : c'est l'écran principal que l'utilisateur voit après sa connexion. J'y ai intégré plusieurs blocs d'information : des accès rapides vers certaines fonctions (badges, inventaire, imprimante et écran badges). La page d'accueil est pensée comme un tableau de bord avec une disposition en grille, des cartes bien délimitées, et des icônes pour guider l'œil.

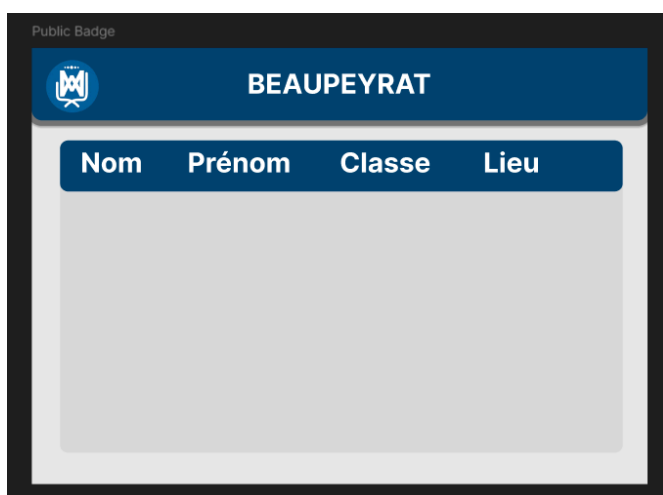


- **Page Inventaire** : cette section liste tous les éléments inventoriés dans l'application (badges, imprimantes, ressources, etc.). J'ai conçu cette page sous forme de tableau, avec des lignes paginées, des colonnes triables, des boutons d'action à droite de chaque ligne (modifier, supprimer), ainsi qu'un bouton d'ajout en haut à

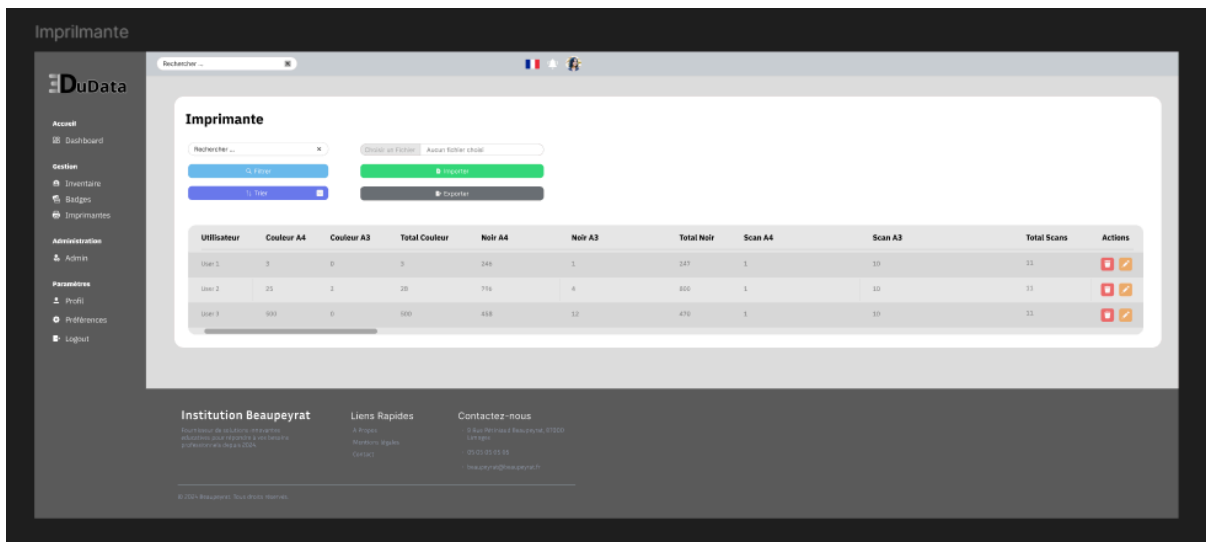
droite. Le tout est structuré pour faciliter la gestion des données et la recherche rapide d'un élément précis.



- **Écran individuel d'un badge** : lorsque l'utilisateur traite un badge, il s'affiche sur cet écran avec l'ensemble des informations liées à ce badge. J'ai organisé cette page en 4 colonnes : le nom, le prénom, la classe et le lieu de retrait



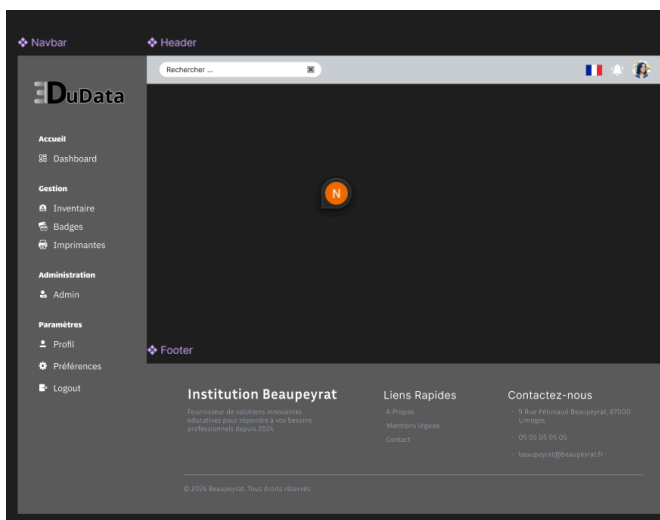
- **Page Imprimante** : cette interface permet de visualiser et gérer les statistiques imprimantes. Elle est construite de façon similaire à l'inventaire, avec une table listant les imprimantes et leurs attributs (copie noir, couleurs, utilisateurs etc.). J'ai également prévu un système de filtres en haut de page, pour faciliter la recherche et la sélection rapide.



2/ Les composants réutilisables

Pour garantir la cohérence visuelle de l'application, j'ai défini plusieurs **composants communs** dès le début de la maquette :

- **La sidebar** : positionnée à gauche, elle contient les liens de navigation vers les pages principales (Accueil, Inventaire, Badges, Imprimantes, etc.). Elle est déployable et visible sur toutes les pages.
- **Le header** : en haut de chaque page il y a la possibilité d'ouvrir ou fermer la sidebar, et l'affichage du nom de chaque page.
- **Le footer** : discret, il est présent en bas de page pour afficher des informations secondaires comme le copyright, le numéro de version, ou des liens vers la documentation.



3/ L'utilité de cette maquette pour le développement

Cette maquette Figma a joué un rôle fondamental dans le bon déroulement du développement frontend. Elle m'a permis :

- de **valider visuellement** les fonctionnalités avant de les coder,
- de **gagner du temps lors de l'intégration**, en suivant un modèle précis pour les tailles, les couleurs, et les espacements,
- de **maintenir une cohérence entre les différentes pages**, car tous les composants étaient prédéfinis,
- d'**itérer rapidement sur l'ergonomie** avant de m'engager dans le code.

Plutôt que de tout improviser au fil du développement, la maquette m'a servi de **référence centrale**, à laquelle je pouvais me référer à chaque nouvelle page ou composant. Elle a aussi facilité la gestion des priorités : une fois toutes les pages listées, j'ai pu définir un ordre d'implémentation logique (login/register → accueil → inventaire → badges → imprimantes), tout en sachant exactement à quoi ressemblerait chaque interface.

2 - Base de donnée logiciel dbdiagram.io / looping

La structuration et la conception de la base de données représentent une étape essentielle dans tout projet web. Dans le cadre du projet **EduData**, j'ai consacré une partie importante de mon travail à la **modélisation relationnelle**, en m'appuyant sur deux outils principaux : **dbdiagram.io** pour la phase de modélisation conceptuelle (MCD/MLD), et **Looping** pour la gestion plus visuelle et technique de la base de données MySQL. Ces outils m'ont permis de réfléchir précisément à la structure des données, aux relations entre entités, aux clés primaires et étrangères, et d'assurer la cohérence du modèle avant son intégration dans Symfony via Doctrine.

Objectif de la modélisation

La base de données est le cœur du projet EduData. Elle stocke l'ensemble des informations : utilisateurs, badges, inventaires, imprimantes, logs, etc. Il était donc primordial de **concevoir une architecture propre, normalisée, évolutive** et adaptée aux fonctionnalités prévues. La phase de modélisation m'a permis :

- de **visualiser l'organisation générale des données** avant leur implémentation technique,
- d'**anticiper les relations** (1-n, n-n) entre entités et les clés de jointure,
- de **valider la structure avec logique métier**,

- de garantir une base **optimisée** pour la gestion des ressources et la performance des requêtes.

Utilisation de dbdiagram.io

J'ai utilisé dbdiagram.io pour réaliser la **modélisation logique de la base de données**. C'est un outil en ligne qui permet de générer des diagrammes de relations entre tables à partir d'un simple langage de description proche du SQL. Grâce à son interface claire, j'ai pu construire progressivement mon schéma de données en visualisant immédiatement les tables, leurs attributs, et les relations.

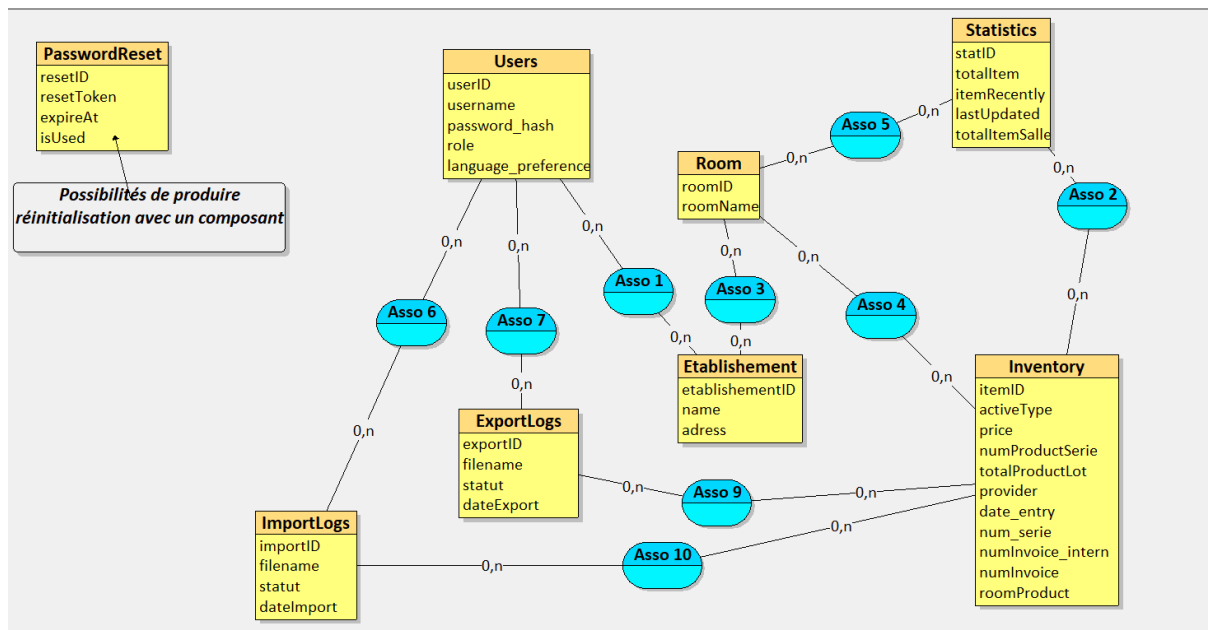
Parmi les avantages de dbdiagram.io :

- **Édition rapide** des structures de tables avec un langage simple,
- Visualisation en **temps réel du schéma relationnel**,
- Export facile au format SQL ou PNG,
- Possibilité de versionner ou de partager le diagramme.

Le diagramme final présente toutes les entités du projet, leurs attributs (avec types et contraintes), les clés primaires, et les relations entre elles via des clés étrangères. Voici quelques-unes des **entités principales** que j'ai définies :

- **Utilisateur** : id, username, roles, password, email, firstname, lastname, .
- **Badge** : id, nom, prenom, date, classe, etat_traitement, lieu.
- **Imprimante** : id, username, total_couleur, total_noir, scan_a4, scan_a3, total_scans, job_charge_count_fcl, job_charge_count_fcs, impression_total_couleur, job_charge_count_mtl, job_charge_count_mts, impression_total_mono, job_charge_count_mcl, job_charge_count_mcs, copie_total_couleur, job_charge_count_mbl, job_charge_count_mbs, copie_total_mono.
- **Inventaire** : id, active_type, price, num_product_serie, total_product_lot, provider, date_entry, num_serie, num_invoice_intern, num_invoice, establishment_id, name_room, reference,

Chaque table a été pensée en suivant les règles de **normalisation**, afin d'éviter les redondances, faciliter les mises à jour, et optimiser les requêtes. Par exemple, les informations des utilisateurs ne sont pas répétées dans les tables liées : seules les clés étrangères permettent de faire les jointures.



Intégration dans Symfony avec Doctrine

La modélisation n'est pas restée théorique : une fois le schéma défini, je l'ai **intégré dans mon application Symfony** à l'aide de **Doctrine ORM**, en créant les entités correspondantes à chaque table. Grâce à la commande `php bin/console make:entity`, j'ai pu créer les classes avec leurs propriétés, types, relations, et contraintes.

Doctrine permet ensuite de **générer automatiquement les migrations SQL** via la commande `make:migration`, puis de les exécuter avec `migrate`. Le lien entre ce que j'avais conçu dans dbdiagram.io et ce qui est déployé dans la base est donc entièrement cohérent.

Cela m'a permis de :

- garder une structure de base **identique à la modélisation initiale**,
- bénéficier des **relations Doctrine** dans le code (annotations `@OneToMany`, `@ManyToOne`, etc.),
- automatiser la création, la mise à jour et la synchronisation de la base,
- travailler avec une **base de données versionnée**, traçable et reproductible.

Conclusion

La phase de modélisation de la base de données a été une étape cruciale dans la réussite du projet EduData. Grâce à **dbdiagram.io**, j'ai pu construire une structure claire, normalisée, et conforme aux besoins du projet. Cette maquette m'a servi de **plan directeur**, aussi bien pour comprendre la logique métier que pour anticiper les requêtes futures.

Avec l'outil **Looping**, j'ai pu mettre en œuvre cette structure concrètement en local, en la testant, en l'ajustant, et en l'exploitant pendant toute la durée du développement. Enfin, l'intégration fluide dans Symfony avec Doctrine m'a permis d'**automatiser la persistance des données**, de **travailler de manière propre et fiable**, et de garder une **parfaite cohérence entre le modèle, la base, et le code**.

Cette méthodologie m'a permis de gagner en rigueur, en efficacité, et surtout de bâtir un socle de données solide, essentiel pour la pérennité et l'évolutivité du projet.

3 - EasyAdmin

Dans le cadre du développement de mon projet EduData, j'ai mis en place une interface d'administration complète afin de permettre une **gestion centralisée et simplifiée des entités principales** de l'application. Pour cela, j'ai choisi d'utiliser le bundle Symfony **EasyAdmin**, une solution performante et rapide à mettre en œuvre qui permet de générer un back-office fonctionnel en quelques lignes de code, tout en restant entièrement personnalisable.

EasyAdmin m'a permis de proposer une interface dédiée aux utilisateurs ayant un rôle administratif, avec un accès sécurisé à diverses sections de gestion (utilisateurs, badges, inventaires, imprimantes, etc.), tout en respectant les bonnes pratiques d'architecture et de sécurité.

Contrôleur principal : **AdminController**

Le fichier **AdminController.php** a pour rôle principal de configurer et structurer l'ensemble du tableau de bord administratif. Voici les trois éléments clés de ce contrôleur :

1. Redirection automatique

Lorsqu'un utilisateur accède à **/admin**, il est automatiquement redirigé vers la page de gestion des utilisateurs (**UserCrudController**) grâce au **AdminUrlGenerator**. Cela garantit une **expérience fluide**, en amenant directement l'administrateur vers une section utile, au lieu de lui laisser une page vide ou générique.

```
#[Route('/admin', name: 'admin')]
public function index(): Response
{
    // return parent::index();

    $routeBuilder = $this->container->get(AdminUrlGenerator::class);
    $url = $routeBuilder->setController(UserCrudController::class)->generateUrl();

    return $this->redirect($url);$routeBuilder = $this->container->get(AdminUrlGenerator::class);
    $url = $routeBuilder->setController(UserCrudController::class)->generateUrl();

    return $this->redirect($url);

    // if(in_array('ROLE_ADMIN', $this->getUser()->getRoles(), false)) {
    //     return new RedirectResponse($this->urlGenerator->generate('login'));
    // }
    // Option 1. You can make your dashboard redirect to some common page of your backend
}
```

3. Structure du menu latéral

Enfin, la méthode `configureMenuItems()` est celle qui définit le menu de navigation latéral de l'administration. Chaque `MenuItem` représente un lien vers une entité gérée par un `CrudController`. Cela permet à l'utilisateur d'administrer les différentes données du projet via une interface graphique intuitive.

Voici quelques exemples d'entités accessibles via le menu :

- Utilisateurs (`User`)
- Établissements (`Establishment`)
- Inventaire (`Inventory`)
- Salles (`Room`)
- Badges (`Badge`)
- Imprimantes (`Imprimante`)
- Logs d'import/export (`ImportLogs`, `ExportLogs`)
- Statistiques (`Statistics`)

Chaque menu est associé à une **icône FontAwesome** pour améliorer la lisibilité et l'ergonomie.

Exemple de configuration CRUD : BadgeCrudController

Pour chaque entité listée dans le menu, un `CrudController` dédié est associé. Ces contrôleurs héritent de la classe `AbstractCrudController` fournie par EasyAdmin, et permettent de personnaliser l'affichage, les champs disponibles, les filtres, les formulaires, etc.

Prenons l'exemple du `BadgeCrudController`, qui permet de gérer les badges distribués aux utilisateurs.

```
public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
    yield MenuItem::linkToRoute('Back to the website', 'fa-solid fa-gear', 'app_dashboard');
    yield MenuItem::linkToCrud('User', 'fa-solid fa-user', User::class);
    yield MenuItem::linkToCrud('Establishment', 'fa-solid fa-building-columns', Establishment::class);
    yield MenuItem::linkToCrud('Inventory', 'fa-solid fa-cart-flatbed', Inventory::class);
    yield MenuItem::linkToCrud('Room', 'fa-solid fa-chair', Room::class);
    yield MenuItem::linkToCrud('ImportsLogs', 'fa-solid fa-file-import', ImportLogs::class);
    yield MenuItem::linkToCrud('ExportLogs', 'fa-solid fa-file-export', ExportLogs::class);
    yield MenuItem::linkToCrud('Statistics', 'fa-solid fa-chart-simple', Statistics::class);
    yield MenuItem::linkToCrud('Badge', 'fa-solid fa-id-card', Badge::class);
    yield MenuItem::linkToCrud('Imprimante', 'fa-solid fa-print', Imprimante::class);
}
```

Avantages de l'approche EasyAdmin

L'utilisation d'EasyAdmin a été un véritable gain de temps et de productivité dans le projet EduData. Voici les principaux bénéfices que j'ai pu en tirer :

- **Interface prête à l'emploi** : sans avoir à développer entièrement un back-office en HTML/CSS/JS, EasyAdmin propose un design propre, responsive, et moderne.
- **Personnalisation facile** : chaque entité peut avoir ses propres champs, ses vues, ses filtres.
- **Sécurité** : l'accès au back-office est protégé et limité aux utilisateurs autorisés (via les rôles Symfony).
- **Rapidité d'intégration** : en quelques lignes de code, j'ai pu gérer l'ensemble des entités critiques du projet.
- **Interopérabilité** : EasyAdmin est parfaitement intégré dans l'écosystème Symfony, ce qui permet d'exploiter tous les services du framework : validation, injections, traductions, événements, etc.

Conclusion

EasyAdmin a constitué un **pilier essentiel dans la partie administration** de mon projet. Il m'a permis de créer un espace de gestion complet et intuitif, utilisé exclusivement par les administrateurs de la plateforme. Grâce à sa structure modulaire, j'ai pu rapidement intégrer les différentes entités du projet, tout en gardant la possibilité de personnaliser l'interface selon les besoins.

Combiné avec les entités Doctrine, les rôles de sécurité Symfony, et une bonne gestion des routes, EasyAdmin s'est révélé être **un outil robuste, fiable et très efficace** pour construire une interface d'administration sans compromis. Cela m'a permis de me concentrer davantage sur la logique métier et les fonctionnalités utilisateurs, tout en garantissant un back-office de qualité professionnelle.

V/ Préparer et déployer une application sécurisée

1 - Codeception

Dans le cadre de mon projet, j'ai choisi d'utiliser Codeception pour réaliser des tests automatisés. Codeception est un framework de test PHP moderne, qui permet d'écrire des scénarios de tests fonctionnels, unitaires ou d'acceptation de manière claire et structurée. Il s'appuie sur une syntaxe en langage naturel, ce qui facilite la compréhension et la maintenance des tests, même pour des développeurs non spécialistes du test logiciel.

J'ai principalement utilisé Codeception pour effectuer des tests d'acceptation, simulant les actions d'un utilisateur sur l'interface web de mon application. Par exemple, un test peut vérifier qu'un utilisateur peut se connecter avec les bons identifiants, ou que le système affiche un message d'erreur en cas de saisie incorrecte. Voici un exemple de test accompagné d'une capture d'écran illustrant son exécution et le résultat obtenu.

```
<?php

class LoginCest
{
    public function loginWithValidCredentials(\AcceptanceTester $I)
    {
        $I->amOnPage('/login');
        $I->fillField('email', 'admin@admin.com');
        $I->fillField('password', 'admin');
        $I->click('Connexion');
        $I->see('Inventaire');
    }
}
```

2 - Déploiement

Modifier le fichier `.env`

Passer l'environnement en production :

```
.env
```

```
APP_ENV=prod
```

Cloner le projet depuis le dépôt Git

```
git clone <url-du-depot>  
cd <nom-du-dossier-cloné>
```

Construire et démarrer les conteneurs Docker

```
docker-compose build  
docker-compose up -d
```

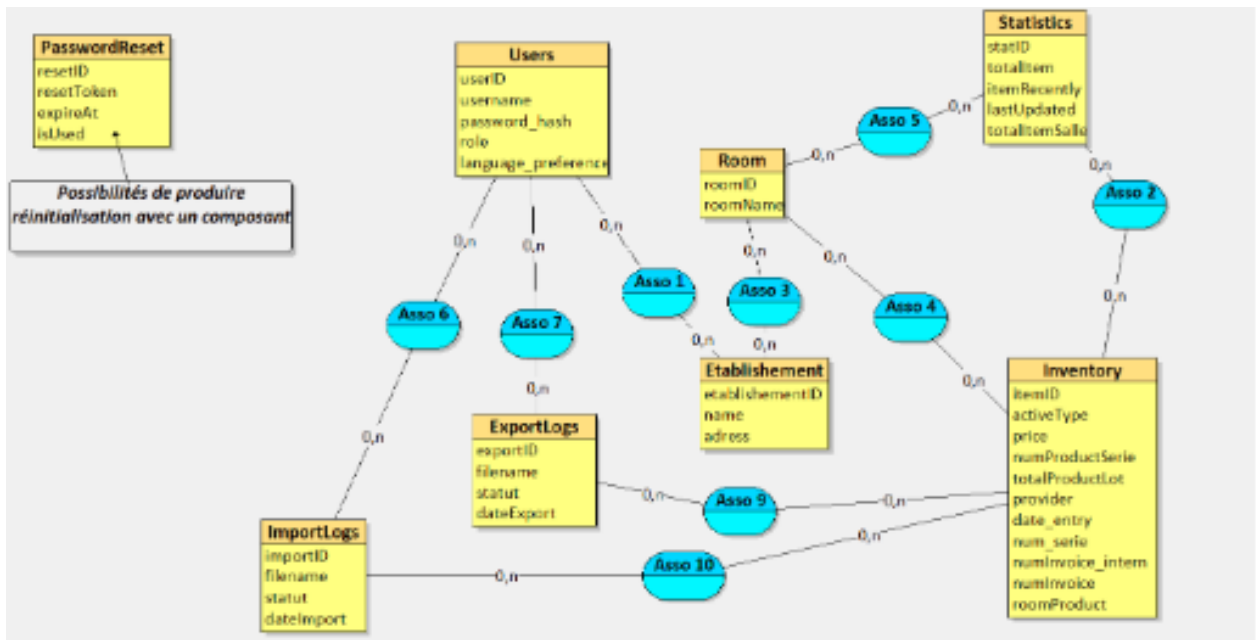
Lancer les migrations de la base de données

```
docker-compose exec php php bin/console doctrine:migrations:migrate
```

Vider et réchauffer le cache Symfony

```
docker-compose exec php php bin/console cache:clear  
docker-compose exec php php bin/console cache:warmup
```


ANNEXES



Search

Inventaire

Label

Choisir un fichier

Aucun fichier choisi

Importer

Exporter

Type Actif	Fournisseur	Date d'arrivée	Número de Série	Número de Facture	Número Facture Interne	Prix Neuf	Número de produit de la série	Nombre total de produits dans le lot	Nom de la Salle	Actions
Serveur	IBM	2024-05-10	ABC12345	INV-2023-001	INT-001	2300	1	4	Info	
Ordinateur	Dell	2024-05-10	ABC12343	INV-2023-001	INT-001	1200	1	1	Prof	
Test	Beaup	2024-05-10	ABC12341	INV-2023-001	INT-001	342	1	3	Test	

Recharger

