|  |
| --- |
| TPI  **Gestion du stock de la section Informatique** |

Alessandro D’Angelo

CIN4B ETML

Chef de projet : Dimitrios Lymberis

Experts : Bernard Oberson, Benjamin Wolf

Lieu : CFPV, Avenue de Valmont 28b, 1010 Lausanne

Date : du 05.05.2023 au 07.06.2023

Durée : 89 heures

**Table des matières**

[1 Analyse préliminaire 3](#_Toc137057459)

[1.1 Introduction 3](#_Toc137057460)

[1.2 Objectifs 4](#_Toc137057461)

[1.3 Planification initiale 4](#_Toc137057462)

[1.4 Méthode de projet 4](#_Toc137057463)

[2 Analyse / Conception 5](#_Toc137057464)

[2.1 Maquette 5](#_Toc137057465)

[2.1.1 Squelette du site 5](#_Toc137057466)

[2.1.2 Authentification & état connexion 5](#_Toc137057467)

[2.1.3 Page d’accueil 6](#_Toc137057468)

[2.1.4 Page de produit 6](#_Toc137057469)

[2.2 Base de données 7](#_Toc137057470)

[2.3 Modèle - Vue – Template 9](#_Toc137057471)

[2.4 Stratégie de test 9](#_Toc137057472)

[3 Réalisation 10](#_Toc137057473)

[3.1 Dossier de réalisation 10](#_Toc137057474)

[3.1.1 Création du projet 10](#_Toc137057475)

[3.1.2 Architecture du projet 12](#_Toc137057476)

[3.1.3 Detail des modèles 14](#_Toc137057477)

[3.1.4 Choix / implémentation des URLs 14](#_Toc137057478)

[3.1.5 Les formulaires en Django 15](#_Toc137057479)

[3.1.6 Système d’authentification 17](#_Toc137057480)

[3.1.7 Restriction selon connexion 18](#_Toc137057481)

[3.1.8 Affichage des tableaux 19](#_Toc137057482)

[3.1.9 Affichage en détail 21](#_Toc137057483)

[3.1.10 Ajout, Modification, Suppression 22](#_Toc137057484)

[3.1.11 Génération QR Code 25](#_Toc137057485)

[3.2 Description des tests effectués 25](#_Toc137057486)

[4 Conclusions 26](#_Toc137057487)

[5 Annexes 27](#_Toc137057488)

[5.1 Table d’illustrations 27](#_Toc137057489)

[5.2 Sources – Bibliographie 27](#_Toc137057490)

[5.3 Planification initiale 28](#_Toc137057491)

[5.4 Planification détaillée 28](#_Toc137057492)

[5.5 Journal de travail 28](#_Toc137057493)

[5.6 Archives du projet 28](#_Toc137057494)

# Analyse préliminaire

## Introduction

Au sein de la filière informatique de l’ETML, il y a un grand nombre de matériaux différents utilisés dans le cadre de la formation des apprentis. Il peut s’agir d’outils portables comme des tablettes, d’outils réseau comme un routeur ou encore de périphériques tels que des écrans. Ces divers matériaux sont labelisés et stockés dans les différents laboratoires de l’établissement et peuvent être rangés dans des armoires.

Le recensement de ces divers équipements est aujourd’hui effectué à l’aide d’un tableau Excel auquel les données y sont rentrées manuellement.

Le but de ce projet est donc de réaliser une application web qui permet une gestion de l’inventaire et des emprunts facilitée.

La technologie choisie pour mettre en place l’application est la combinaison du langage Python avec le framework Django, qui permet la réalisation d’application web.

*Ce chapitre décrit brièvement le projet, le cadre dans lequel il est réalisé, les raisons de ce choix et ce qu’il peut apporter à l’élève ou à l’école. Il n’est pas nécessaire de rentrer dans les détails (ceux-ci seront abordés plus loin) mais cela doit être aussi clair et complet que possible (idées de solutions). Ce chapitre contient également l’inventaire et la description des travaux qui auraient déjà été effectués pour ce projet.*

Ce projet se déroule dans le cadre d’un Travail Pratique Individuel (TPI) et fait suite au projet d’approfondissement qui utilisait les mêmes technologies.

L’environnement de développement utilisé est le suivant :

* Un MacBook Air 2020 sous MacOS 13.2.1 Ventura
* Visual Studio Code
* La suite Microsoft Office 365
* Un serveur Python Django
* Un dépôt GitHub : [github.com/alessdangelo/tpi\_gestock](https://github.com/alessdangelo/tpi_gestock)

Les modules ayant été utile à la réalisation de ce projet :

* [302](https://www.modulbaukasten.ch/module/302/3/fr-FR?title=Utiliser-les-fonctions-avanc%C3%A9es-d%E2%80%99Office) : Utilisation avancée d’office
* [104](https://www.modulbaukasten.ch/module/104/3/fr-FR?title=Impl%C3%A9menter-un-mod%C3%A8le-de-donn%C3%A9es), 105, 151 : Schématiser, implémenter et gérer une base de données
* [306](https://www.modulbaukasten.ch/module/306/4/fr-FR?title=R%C3%A9aliser-de-petits-projets-dans-son-propre-environnement-professionnel) & [426](https://www.modulbaukasten.ch/module/426/1/fr-FR?title=D%C3%A9velopper-un-logiciel-avec-des-m%C3%A9thodes-agiles) [431](https://www.modulbaukasten.ch/module/431/2/fr-FR?title=Ex%C3%A9cuter-des-mandats-de-mani%C3%A8re-autonome-dans-son-propre-environnement-professionnel): Gestion de projet
* 120, 318, & [411](https://www.modulbaukasten.ch/module/411/1/fr-FR?title=D%C3%A9velopper-et-appliquer-des-structures-de-donn%C3%A9es-et-algorithmes) : Développer et appliquer des structures de données et algorithmes

## Objectifs

Selon le cahier des charges les points suivants doivent être présents :

* L’interface s’adapte en fonction de la connexion (menu personnalisé, profil utilisateurs)
* Il est possible d’ajouter, modifier, supprimer un article et visualiser le détail de celui-ci.
* Il est possible d’ajouter, modifier, supprimer un emprunt et visualiser le détail de celui-ci
* Il est possible d’afficher une liste d’articles avec filtre et vision si l’article est emprunté
* Affichage sous forme de liste de tous les articles empruntés actifs et ceux de l’utilisateur connecté. Les deux listes triées selon la date d’emprunt plus récente.
* Génération d’un QR code pour un article et impression selon un format réglable
* La modélisation de la base de données qui respecte la nomenclature Merise et le MCD / MLD / MPD sont présents et corrects

## Planification initiale

La planification initiale est fournie en [annexe](#_Annexes).

## Méthode de projet

La méthode des 6 pas a été utilisée, l’agilité n’étant pas nécessaire dans ce projet. Elle consiste en 6 points définis comme tels :

1. **S’informer**

La première étape se focalise sur la compréhension des tâches, définir les objectifs et besoins du projet.

1. **Planifier**

Phase de planification initiale et détaillée.

1. **Décider**

Phase de prise de décision selon le cahier des charges.

1. **Réaliser**

Étape de réalisation des tâches définies.

1. **Contrôler**

Tester le bon fonctionnement de l’application et de son implémentation selon les directives du cahier des charges.

1. **Évaluer**

Passer en revue ce qui a été réalisé avec le chef de projet et les experts.

# Analyse / Conception

## Maquette

La maquette permet d’avoir une vue indicative de la disposition des différents éléments dont ont besoin chaque page afin d’anticiper la structure du code.

### Squelette du site

Le menu, le bandeau ainsi que le pied de page restent les mêmes à travers les pages, seule la partie de contenu change.

*Bandeau*

*Bloc menu*

*Pied-de-page*

*Contenu*

Figure 1 - Squelette du site

### Authentification & état connexion

A screenshot of a login form

Description automatically generated with low confidenceEn haut à gauche de chaque page, dans le bandeau, est présent un état de connexion. Si un utilisateur est connecté, il sera affiché son nom d’utilisateur, son nombre d’emprunts en cours ainsi que d’un bouton de déconnexion. Si personne n’est connecté, seul un bouton de connexion s’affiche.



### A screenshot of a computer Description automatically generated with medium confidencePage d’accueil

Figure 2 - Maquette de l'accueil

Une simple maquette de l’accueil réalisée afin d’avoir une idée globale sur le contenu du site, les pages ayant soit des informations, des tables ou un formulaire.

### A screenshot of a computer Description automatically generated with low confidencePage de produit

Figure 3 - Maquette de détail

Idée initiale d’une page détaillée. Pour une question de temps à disposition, ce style n’a pas été implémenté en faveur d’un style plus simple et tout aussi fonctionnel.

## Base de données

*A picture containing text, screenshot, diagram, font

Description automatically generated*Pour la base de données, Le MCD à été schématisé à l’aide de DB-Main afin d’avoir une idée visuelle et réfléchie des differentes entités et des possibles relations entre elles :

Figure 4 - Schéma MCD

t\_products :

Elle contient un article qui fait partie d’une catégorie faisant elle-même partie d’un type . Un produit est doté d’un nom, d’une image, d’une note/description ainsi que d’un prix et d’une date d’achat.

t\_types :

Contient le nom d’un type (exemple : Réseau).

t\_categories :

Contient le nom d’une catégorie (exemple : Routeur).

t\_article :

Un article est un produit ayant été labelisé (ici stocké dans « artName ») et qui peut être en premier lieu situé dans un labo et a la possibilité d’être dans une armoire de ce dernier. Un article peut être emprunté par un utilisateur, auquel cas les informations suivantes seront enregistrées :

* Un emplacement de location (Exemple : maison ou office)
* La date de l’emprunt
* La date de retour de l’emprunt
* Une note liée à l’emprunt (exemple : État de l’article au moment de l’emprunt)

t\_user :

Ici sera stocké le nom de l’utilisateur, s’il est administrateur et s’il est toujours actif.

t\_room :

Contient le nom d’une salle/labo (exemple : A12) qui peut être ou ne pas être doté d’une ou plusieurs armoires.

t\_cupboard :

Le nom d’une armoire (exemple : ARM-201) qui est obligatoirement située dans une salle.

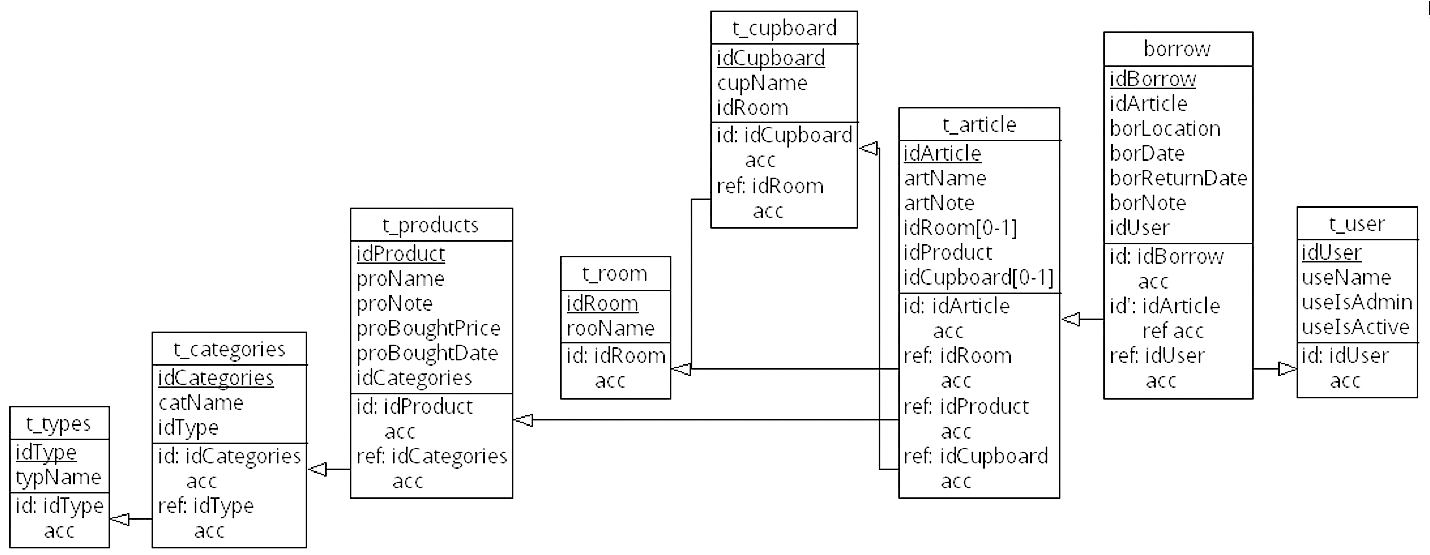
Le schéma MCD a ensuite été converti en MLD afin de visualiser les relations entre les tables :

Figure 5 - Schéma MLD

Le schéma MLD reste nonobstant semblable au MCD, seule la relation d’emprunt ‘borrow’ s’est transformée en table, ici ‘t\_borrow’.

## Modèle - Vue – Template

Pour ce projet, une architecture similaire au Modèle - Vue – Contrôleur (MVC) mais propre à Django a été utilisé : Le Modèle - Vue – Template (MVT). La Vue du MVT diffère, elle agit plutôt comme un contrôleur, qui irait ensuite remplir le fichier Template, qui serait donc l’équivalent de la vue du MVC. Voici un schéma réalisé détaillant le pattern MVT :

Graphical user interface

Description automatically generated

Figure 6 - Schéma MVT

1. L’utilisateur envoie une requête HTTP au serveur Django.
2. Le serveur Django reçoit la requête et détermine quelle vue gère cette requête
3. La vue corresspondante est appelée et traite la logique associée à la requête (Communication avec la base de données depuis le model , résolutions de formulaires, …)
4. La vue remplit ensuite la page HTML demandée.

## Stratégie de test

Il est important de procéder à des tests afin de s’assurer du bon fonctionnement et ne pas détériorer l’expérience utilisateur.  
Afin d’organiser les tests, ils seront divisés comme tels :

* Le nom de la page du site où le test sera effectué
* Description du test à effectuer
* Le résultat attendu
* Le résultat obtenu
* Temps requis pour corriger et/ou finaliser

Les tests seront réalisés manuellement au fur et à mesure du développement, avant de faire une vérification à la fin du projet.

# Réalisation

## Dossier de réalisation

### Création du projet

Pour instancier notre application Django, il suffit de quelques opérations.

Création de la structure du projet :

django-admin startproject tpi\_gestock

Création de notre application en elle-même, ici appelée « app\_gestock »

python3 manage.py startapp app\_gestock

Afin d’avoir accès au panel administrateur, la commande suivante est utilisée pour nous créer un compte utilisateur aux droits élevés :

python3 manage.py createsuperuser

Création d’un dossier “templates” dans le dossier de l’application qui contient les pages html requises. Un fichier « base.html » y est ajouté, fonctionnant comme fichier de base, sont contenu sera remplacé par d’autres pages.

*base.html*

{% load static %}

{% comment %} Base HTML file used as a base for any HTML page {% endcomment %}

<!DOCTYPE html>

<html lang="fr">

<head>

<meta charset="UTF-8" />

<title>[title]</title>

<link rel="stylesheet" href="{% static 'css/style.css' %}" />

<link

rel="icon"

type="image/x-icon"

href="{% static 'resources/favicon.svg' %}"

/>

</head>

<body>

{% comment %} Block will be replaced with content from other pages{% endcomment %}

{% block content %}

{% endblock %}

</body>

</html>

Ici, la page d’index remplace le bloc « content » de base.html :

*index.html*

{% extends 'base.html' %}

{% block title %}Vue d'ensemble{% endblock %}

{% block content %}

# Boutons d’ajout

<a href="{% url 'addProduct' %}">Ajouter un produit</a>

<a href="{% url 'addArticle' %}">Ajouter un article</a>

# Table d’articles

<h2 Liste d'articles</h2>

<table id="articles-table">

<thead>

<tr>

<th>Type</th>

<th>Categorie</th>

<th>Product</th>

<th>Article Name</th>

<th>Emprunté par</th>

<th>Description</th>

<th>Lieu de location</th>

<th>Date d'emprunt</th>

<th>Date de retour</th>

</tr>

</thead>

</table>

# Table de produits

<h2>Liste de produits</h2>

<table id="products-table">

<thead>

<tr>

<th>Type</th>

<th>Categorie</th>

<th>Produit</th>

<th>Description</th>

</tr>

</thead>

</table>

{% endblock content %}

### Architecture du projet

A screenshot of a phone

Description automatically generated with low confidenceLes dossiers et fichiers sont organisés comme suit, selon l’architecture Django par défaut

Figure 7 - Architecture de projet

Ici ‘app\_gestock’ est l’application créé dans le projet (plusieurs applications peuvent être crées dans un seul projet)

‘templates’ est un dossier créé manuellement mais qui est géré et reconnu par Django pour les fichiers HTML correspondant au visuel

‘media’ contient les images présentes dans l’application

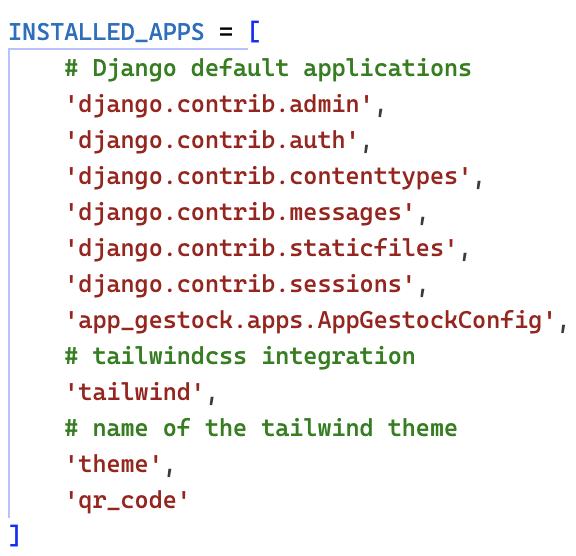
‘theme’ contient les fichiers de style de TailwindCSS

‘tpi\_gestock’ correspond au projet qui contient notre application. Ici se trouvent les paramètres globaux.

‘db.sqlite3’ est la base de données SQLite intégrée à Django

**3.2 Paramétrage**

Les configurations du projet sont effectuées dans le fichier 'settings.py'. Après création de l'application et avant de commencer à coder, il est nécessaire de déclarer la création de l'application dans la constante qui gère les applications reconnues par Django.



*settings.py*

Toutes ces applications sont par défaut pris en compte par Django lors de la création de notre projet.

La déclaration de l’intégration de TailwindCSS a été rajouté ainsi que de ‘qr\_code’ pour la génération de codes QR.

Il suffit maintenant d’entrer la commande suivante afin de pouvoir utiliser le framework tailwind :

pip install django-tailwind

Puis cette commande lors du développement de l’application afin de prendre en compte les derniers changements de tailwind :

python manage.py tailwind start

### Detail des modèles

Les Modèles en Django sont nécessaires au bon fonctionnement du projet car ceux-ci représentent la structure de la base de données et permet d’interagir avec cette dernière. Notez que la base de données est ici en SQLite, base de données générée et utilisée par défaut par Django.

Les Modèles sont situés dans un fichier nommé ‘models.py’, comme détaillé ci-dessus.

Voici un exemple de modèle :

*models.py*

class t\_products(models.Model):

"""

Table representing a product.

"""

idProduct = models.AutoField(primary\_key=True) # Unique id for the product.

proName = models.CharField(max\_length=75, unique=True) # Name of the product.

proNote = models.CharField(max\_length=250) # Note for the product.

proImage = models.CharField(max\_length=500) # Image link of the product.

proBoughtPrice = models.FloatField() # Price at which the product was bought.

proBoughtDate = models.DateField() # Date on which the product was bought.

# Foreign keys to the corresponding tables

fkArticle = models.ForeignKey(t\_article, on\_delete=models.CASCADE, null=True)

fkCategory = models.ForeignKey(t\_categories, on\_delete=models.CASCADE)

fkType = models.ForeignKey('t\_types', on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return self.proName # Return the name instead of the object itself

### Choix / implémentation des URLs

Concernant les URLs, Django crée par défaut dans le projet un fichier nommée ‘urls.py’, ici il a été décidé pour rendre le code plus synthétique et plus distingué de mettre les URLs de l’application seulement dans celle-ci et pas dans le projet, c’est pourquoi les URLs du projet ressemble à cela :

*urls.py du projet*

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('app\_gestock.urls')), # Get urls from the app : app\_gestock

]

Seul le lien pour la configuration de l’interface admin de django est laissé ici car c’est celui du projet. L’autre lien inclus le second fichier d’URLs, celui créé dans l’application.

Dans l’application, ce fichier nommé également ‘urls.py’ remplis la fonction principale de celle-ci, à savoir rediriger un lien d’une requête, sur une vue de ‘views.py’ menant sur la page correspondante ([voir détail MVT](#_Modèle_Vue_Template)).

*urls.py de l’application*

from . import views

# Import the path & include function from the django.urls module

from django.urls import path, include

# Define a list of URL patterns for the web application

urlpatterns = [

# Map the URLs to it's view function and give it a name

path('', views.index, name='index'),

path('product', views.product, name='product'),

path('article', views.article, name='article'),

path('borrow', views.borrow, name='borrow'),

path('borrows', views.borrows, name='borrows'),

path('products', views.products, name='products'),

path('articles', views.articles, name='articles'),

path('addProduct', views.addProduct, name='addProduct'),

path('editProduct', views.editProduct, name='editProduct'),

path('deleteProduct', views.deleteProduct, name='deleteProduct'),

path('addArticle', views.addArticle, name='addArticle'),

path('editArticle', views.editArticle, name='editArticle'),

path('deleteArticle', views.deleteArticle, name='deleteArticle'),

path('addBorrow', views.addBorrow, name='addBorrow'),

path('editBorrow', views.editBorrow, name='editBorrow'),

path('jsondata', views.jsondata, name='jsondata'),

# Include the authentication URLs provided by Django and prefix them with /accounts/

path('accounts/', include('django.contrib.auth.urls')),

]

### Les formulaires en Django

Le Framework Django possède deux implémentations possibles de formulaires : forms.Form et forms.ModelForm. Ce sont deux classes qui permettent de définir et de valider des formulaires HTML.

Le premier intitulé « forms.Form» est une classe de formulaire de base plus classique qui ne communique pas directement avec le modèle et où il faut définir manuellement chaque champ et ses règles de validation.

D'autre part, forms.ModelForm est une sous-classe de forms.Form qui fournit une génération automatique de formulaire basée sur le modèle.

forms.Form et forms.ModelForm fournissent toutes deux des méthodes pour valider les données du formulaire et rendre le formulaire en HTML.

Ici, modelForm a été utilisé afin de garantir une simplification du code et de la communication avec le modèle.

Voici le formulaire d’ajout et de modification d’un article :

*forms..py*

class articleForm(forms.ModelForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

class Meta:

model = t\_article

fields = ['artName', 'artNote','fkRoom','fkProduct', 'fkCupboard']

labels = {

'artName': 'Label',

'fkProduct': 'Produit',

'artNote': 'Note',

'fkRoom': 'Labo',

'fkCupboard': 'Armoire'

Dans le formulaire sont définis les champs requis présents dans le modèle avec le label correspondant.

Formulaire d’emprunt :

*borrow.py*

class borrowForm(forms.ModelForm):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['borDate'].initial = datetime.date.today()

self.fields['borReturnDate'].initial = datetime.date.today() + datetime.timedelta(days=30)

class Meta:

model = t\_borrow

fields = ['borLocation', 'borDate', 'borReturnDate', 'borUser', 'fkArticle']

labels = {

'borLocation': 'Lieu demprunt :',

'borDate': 'Date demprunt :',

'borReturnDate': 'Date de retour :',

'borUser': 'Emprunté par :',

'fkArticle': 'Article :',

}

widgets = {

'borDate': DateInput(format=('%Y-%m-%d'),attrs={'type': 'date', 'class': 'dateForm'}),

'borReturnDate': DateInput(format=('%Y-%m-%d'),attrs={'type': 'date', 'class': 'dateForm'}),

}

Nous définissons tout d’abord une date initiale d’emprunt, la date actuelle. La date de retour, elle, est par défaut 30 jours après l’emprunt.

Les champs et leurs labels sont aussi présents. La partie widget permet, comme son nom l’indique, l’utilisation d’un widget de choix de date.

### Système d’authentification

Ici, seule la fonctionnalité de connexion est nécessaire. L’ajout d’utilisateur se fait dans le panel administrateur. Il serait cependant possible d’ajouter les fonctionnalités restantes, l’implémentation utilisant le système fourni par Django.

Dans le fichier ‘urls.py’, est ajouté le lien menant à la connexion :

*urls.py de l’application*

  # Include the authentication URLs provided by Django and prefix them with /accounts/

path('accounts/', include('django.contrib.auth.urls')),

La connexion faisant appel à la méthode de connexion de Django, il suffit de rajouter les pages HTML de connexion et de déconnexion :

*login.html*

{% if form.errors %} {# Check if there are errors in the form #}

<p>Your username and password didn't match. Please try again.</p>

{% endif %}

<form method="post" action="{% url 'login' %}"> {# Login form #}

{% csrf\_token %}

<table>

<tr>

<td>{{ form.username.label\_tag }}</td>

<td>{{ form.username }}</td>

</tr>

<tr>

<td>{{ form.password.label\_tag }}</td>

<td>{{ form.password }}</td>

</tr>

</table>

<input type="submit" value="login">

<input type="hidden" name="next" value="{{ next }}">

</form>

*logout.html*

<p>Logged out!</p>

<a href="{% url 'login'%}">Click here to login again.</a>

### Restriction selon connexion

Si l’utilisateur n’est pas connecté sur le site, il n’aura accès qu’à la page d’accueil, redirigeant sur la page de connexion lors de tentative d’accès sur les autres pages si l’utilisateur n’est pas authentifié.

Pour cela, la simple ligne ‘@login\_required’ est nécessaire au-dessus de chaque vue.

*views.py*

@login\_required

def products(request):

menuTypes = dynMenu()

context={'menuTypes': menuTypes, 'borTotal': getTotalUserBorrows(request)}

return render(request, 'products.html', context)

Le bandeau de la page d’accueil est modulable, affichant un bouton de connexion si personne n’est identifié et dans le cas contraire, un bouton de déconnexion accompagné de son nom et son nombre d’emprunts actuel.

*base.html*

{% if user.is\_authenticated %}

<div>

<p>{{request.user.username}}</p>

<p>Emprunts en cours : {{ borTotal }}</p>

<a href="{% url 'logout' %}" href="{% url 'logout' %}">Déconnexion</a>

A picture containing text, font, white, design

Description automatically generated</div>

{% else %}

<a href="{% url 'login' %}">Connexion</a>

{% endif %}

Figure 8 - Module d'état de connexion

### Affichage des tableaux

Une vue en tableaux étant nécessaire, il a été choisi par question de temps et de simplicité d’utiliser un plug-in de base de JQuery nommé ‘DataTables’

#### Récupération des données

Afin de passer les données aux tableaux, il est nécessaire de passer par une requête JSON. La première étape consiste à récupérer les données souhaitées d’une table dans la base de données. Un paramètre GET contenant le nom de la table est récupéré, avant de passer dans une condition ‘if’. Les données sont envoyées à l’aide du module ‘JsonResponse’, importé au préalable au début du fichier.

Voici un exemple utilisant cette url:

http://127.0.0.1:8000/jsondata?table=t\_products

*views.py*

table = request.GET.get('table') # On récupère la table demandée

if table == 't\_products': # Chercher les données de la table de produits

data = list(t\_products.objects.select\_related(

'fkType',

'fkCategory'

).values(

'fkType\_\_typName',

'fkCategory\_\_catName',

'proName',

'proNote',

'idProduct',

))

return JsonResponse(data, safe=False) # Envoi des données JSON

Un exemple de données récupérées :

"fkType\_\_typName":"Accessoires",

"fkCategory\_\_catName":"Moniteur",

"proName":"Thinkvision 27\"",

"proNote":"2K Resolution, 60FPS ",

"idProduct":2

#### Création des tableaux

*products.html*

Pour le fonctionnement de la table, il faut en premier lieu l’instancier en HTML :

<table id="products-table" >

<thead>

<tr>

<th class="px-2 py-2">Type</th>

<th class="px-2 py-2">Categorie</th>

<th class="px-2 py-2">Produit</th>

<th class="px-2 py-2">Description</th>

</tr>

</thead>

</table>

Puis en Javascript/JQuery :

*products.html*

$(document).ready(function() {

$('#products-table').DataTable({

scrollX: true, // permet le défilement horizontal

ajax: {

url: '{% url 'jsondata' %}?table=t\_products', // source des données

type: 'GET',

dataSrc: ''

},

columns: [ // Champs des données

{ data: 'fkType\_\_typName' },

{ data: 'fkCategory\_\_catName' },

{ data: 'proName' },

{ data: 'proNote' },

],"rowCallback": function(row, data) {

$(row).on('click', function() {

// Permet d’accéder aux détails d’un produit lors d’un clique

window.location.href = '/product?id=' + data.idProduct;

A screenshot of a computer

Description automatically generated with low confidenceLe résultat final :

Figure 9 - Tableau de produits

L’implémentation des autres tableaux sont tous similaires.

#### Couleurs selon état

Selon l’état d’un article, son affichage dans les tables change. Dans la page d’article un article disponible s’affiche en vert. S’il ne l’est pas, il est en rouge. Les articles empruntés par l’utilisateur s’affichent en bleu dans la page des emprunts.



Figure 10 - Tableau d'articles

A screenshot of a computer

Description automatically generated with low confidence

Figure 11 - Tableau d'emprunts

### Affichage en détail

Il est possible d’afficher les détails d’un produit, d’un article et d’un emprunt. Leur vue ne diffère que très peu ;

* Ils partagent tous un nom de produit, une image, ainsi qu’une note/description. Il y a possibilité de modification ou de suppression sur chacune de ces pages.
* Un produit contient en plus une date et prix d’achat, une catégorie ainsi qu’un type.
* Un article est doté d’un label, d’un labo et/ou une armoire ainsi qu’un QR code contenant ces informations.
* Un emprunt contient un label, un lieu de location et l’emprunteur.

A screen shot of a computer monitor

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceA qr code with a screen

Description automatically generated with low confidence

Figure 12 - Détail d'emprunt

Figure 13 - Détail de produit

Figure 14 - Détail d'article

### Ajout, Modification, Suppression

Un seul formulaire ‘modelForm’ est nécessaire pour l’ajout, modification et suppression des données. Il est cependant nécessaire de séparer les formulaires d’article, de produit ainsi que d’emprunt.

Les pages HTML utilisent le même code pour l’affichage du formulaire :

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Sauvegarder</button>

</form>

#### Ajout

A l’accueil du site se trouve deux boutons d’ajout, un destiné aux produits et l’autre aux articles.

Lorsqu’un bouton est pressé, l’utilisateur est redirigé sur la vue correspondante.

*views.py*

def addArticle(request):

if request.method == 'POST': # Si le formulaire a déjà été envoyé

form = articleForm(request.POST)

if form.is\_valid():

# Sauvegarde dans la base de données uniquement si formulaire OK

form.save()

id = t\_article.objects.latest('idArticle').idArticle

# Redirection sur la page détaillée du produit

return HttpResponseRedirect(f'article?id={id}')

else:

form = articleForm()# Instanciation initiale du formulaire

return render(request, 'addArticle.html', {'form': form})

L’ajout comprend un formulaire avec les champs nécessaires.

*forms.py*

class articleForm(forms.ModelForm):

class Meta:

model = t\_article // Source des données

fields = ['artName', 'artNote','fkRoom','fkProduct', 'fkCupboard'] // Champs

labels = { // Label des champs

'artName': 'Label : ',

'fkProduct': 'Produit : ',

'artNote': 'Note : ',

'fkRoom': 'Labo : ',

'fkCupboard': 'Armoire : '

}

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

Figure 15 - Ajout de produit

Figure 16 - Modification de produit

L’ajout d’un emprunt ce fait cependant depuis la vue d’un article uniquement si celui-ci est disponible à l’emprunt. Si ce n’est pas le cas, le bouton d’emprunt ne s’affichera pas.

A qr code with a screen

Description automatically generated with low confidence

Figure 17 - Détail d'article

Pour l’image d’un produit, il a fallu tout d’abord ajouter le module python ‘Pillow’ qui permet la gestion d’image.

pip install pillow

Dans les parametres du projet, il faut rajouter l’emplacement de tous types de médias, ici ils seront placés dans un dossier ‘media’, à la racine du projet.

*settings.py*

MEDIA\_URL = '/media/'

MEDIA\_ROOT = BASE\_DIR / 'media'

Ceci fait, il faut maintenant changer le champ d’image dans le modèle en tant que « imagefield ».

*model.py*

proImage = models.ImageField(upload\_to="proImages", blank=True, null=True)

‘upload\_to’ est l’emplacement des images. Ici il sera placé dans un dossier ‘proImages’ dans le dossier ‘medias’

#### Modification

Pour modifier un produit, un article ou un emprunt, il faut avant tout récupérer son identifiant unique, Ici réalisé à l’aide d’une requête GET et de l’url fournie.

Voici un exemple utilisant cette url:

http://127.0.0.1:8000/editArticle?id=1

*views.py*

def editArticle(request):

id = request.GET.get('id') # Identifiant d’article

# Récupération article lié à l’ID

article = get\_object\_or\_404(t\_article, idArticle=id)

if request.method == 'POST': # Si le formulaire a déjà été envoyé

form = articleForm(request.POST, instance=article)

if form.is\_valid():

# Sauvegarde dans la base de données uniquement si formulaire OK

form.save()

# Redirection sur la page détaillée du produit

return HttpResponseRedirect('/article?id='+id+'')

else:

form = articleForm(instance=article)

return render(request, 'editArticle.html', {'form': form})

#### Suppression

La suppression est similaire à la modification, l’id est recupéré à travers une requête GET mais ici pas d’interaction avec les formulaires ; L’objet a supprimé est récupéré à l’aide de l’ID avant d’être supprimé avec sa fonction de suppression : « delete() ». L’utilisateur est ensuite redirigé sur la page d’accueil.

Voici un exemple utilisant cette url:

http://127.0.0.1:8000/deleteArticle?id=1

*views.py*

def deleteArticle(request):

id = request.GET.get('id') # Identifiant d’article

# Récupération article lié à l’ID

article = get\_object\_or\_404(t\_article, idArticle=id)

article.delete() # Suppression de l’article

return HttpResponseRedirect('/')

Toute suppression s’effectue en cascade et supprime ce qui est nécessaire. Dans le cas de suppression d’un article emprunté, les deux sont donc effacés, un emprunt ne pouvant pas exister sans article.

### Génération QR Code

Il faut d’abord s’assurer que le module est installé :

pip install django-qr-code

Ceci fait, il est nécessaire de récupérer les données d’un article

*views.py*

qr\_content = f"ETML - Informatique - {article.artName} - {article.fkProduct.proName} - {article.artNote} - {article.fkRoom} / {article.fkCupboard}"

Cette variable est ensuite utilisée dans le fichier HTML pour générer les QR Code à l’aide du module, après l’avoir chargé au début du fichier :

*article.html*

{% load qr\_code %}

{% qr\_from\_text qr\_content size=8 version=20 error\_correction="Q" %}

## Description des tests effectués

| Fonctionnalité | Test | Impact en cas d’échec | Résultat attendu | Résultat obtenu |
| --- | --- | --- | --- | --- |
| jsondata | Accès non authentifié | Risque des données car elle sont visible à tous | Données inaccessible si l’utilisateur n’est pas authentifié | **OK - 100%** |
| deleteBorrow | Suppression des emprunts | Risque de perte des données | Un emprunt uniquement est supprimé | **OK - 100%** |

# Conclusions

Dans sa finalité, le rendu de projet remplit les objectifs techniques demandés. Ce projet avec des technologies peu communes au sein de l’ETML était très stimulant de très bons sentiments positifs en ressort :

* L’utilisation de formulaire facilitée grâce à Django
* Projet fonctionnel
* Aucune surprise sur les résultats obtenu des tests effectués

Il est néanmoins important de relever que les tableaux ont pu causer une perte de temps non négligable car il aurait fallut soit l’implémenter de zéro, soit utiliser une librairie mais qui n’a pas forcément été créé pour Django,

Le projet peut encore évoluer en ajoutant notamment un système utilisateur complet , ajouter d’autre types de produits tel que des meubles.

Bilan de planification :

Concernant la planification, un certain retard a pu être observé au ¾ du projet a cause de l’implémentation de la vue des données en tableau qui fût sous-estimée, ayant affaire à un plug-in inconnu. Le reste du déroulement projet était cependant fluide et je suis certain que sans ce problème de tableaux, il se serait déroulé sans encombre.

A picture containing text, screenshot, number, font

Description automatically generated

# Annexes

## Table d’illustrations

Situation de départ :

Le but de ce projet était de proposer un système de gestion d’articles, de produits ainsi que d’emprunts dans un établissement ou le besoin de technologie est constant, pouvoir garder trace de l’emplacement des divers articles.

Mise en œuvre :

Le programme devait permettre une connexion utilisateur, une gestion complète des produits, des articles ainsi que d’emprunts. Il est possible d’afficher ces informations de manière détaillée ou sous forme de tableaux. Un QR Code doit être généré pour chaque article.

Résultats :

La totalité des points techniques demandés ont pu être réalisés et l’application est fonctionnelle.

Table d’illustrations

[Figure 1 - Squelette du site 5](#_Toc137046019)

[Figure 2 - Maquette de l'accueil 6](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046020)

[Figure 3 - Maquette de détail 6](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046021)

[Figure 4 - Schéma MCD 7](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046022)

[Figure 5 - Schéma MLD 8](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046023)

[Figure 6 - Schéma MVT 9](#_Toc137046024)

[Figure 7 - Architecture de projet 12](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046025)

[Figure 8 - Module d'état de connexion 18](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046026)

[Figure 9 - Tableau de produits 20](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046027)

[Figure 10 - Tableau d'articles 21](#_Toc137046028)

[Figure 11 - Tableau d'emprunts 21](#_Toc137046029)

[Figure 12 - Détail d'emprunt 21](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046030)

[Figure 13 - Détail de produit 21](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046031)

[Figure 14 - Détail d'article 21](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046032)

[Figure 15 - Ajout de produit 23](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046033)

[Figure 16 - Modification de produit 23](file:////Users/aless/Documents/Dev/GitHub/tpi_gestock/docs/TPI-aledangelo_gestock.docx#_Toc137046034)

[Figure 17 - Détail d'article 23](#_Toc137046035)

## Sources – Bibliographie

<https://django-qr-code.readthedocs.io/en/latest/pages/README.html>

<https://datatables.net/>

<https://tailwindcss.com/docs/hover-focus-and-other-states>

<https://tailwindcss.com/>

<https://docs.djangoproject.com/en/4.2/>

<https://youtu.be/CTrVDi3tt8o>

<https://docs.djangoproject.com/en/4.1/topics/auth/default/>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/skeleton_website>

## Planification initiale

Voir document 1 en annexe

## Planification détaillée

Voir document 2 en annexe

## Journal de travail

Voir document 3 en annexe

## Archives du projet

<https://github.com/alessdangelo/tpi_gestock>