

FICHE DE PROCEDURE

Cette fiche de procédure va décrire la procédure d'installation et de déploiement de notre application web.



SAÉ2.03 :
Mettre en
place une
solution
informatique
pour
l'entreprise

Sommaire :

1) Objectif du projet.....	1
2) Diagramme de Gantt et Planning	2
3) Schéma relationnel du projet.....	3
4) Création du projet Django pour notre projet	5
5) Base de données MySQL.....	5
6) Déploiement de notre projet	8
7) Fonctionnement du site	11
8) Apparence du site avec HTML et CSS	14
9) Conclusion du projet / SAÉ	16

1) Objectif du projet

Le projet donné pour cette **SAÉ2.03**, consiste à développer un site web permettant de gérer les **absences** d'un département scolaire ou d'une université en fonction des différents cours, avec un système permettant d'ajouter des justificatifs d'absences par la suite. Le schéma de données comprend les entités suivantes :

- **Des groupes d'étudiants (id, nom)**
- **Des étudiants (id, nom, prénom, email, groupe, photo)**
- **Des enseignants (id, nom, prénom, email)**
- **Des cours (id, titre du cours, date, enseignant, durée, groupe)**
- **Des absences à des cours (étudiants, cours, justifié ou non, justification)**

Il faudra donc, pour ce projet, mettre en place un **CRUD** (*Create, Read, Update, Delete*) pour chaque type de données énoncées ci-dessus.

Notre site Web, que nous devons mettre en place, devra permettre la saisie de nouveaux cours et les absences liées à ce cours. Nous devons également être en mesure de vérifier / valider les absences et d'insérer des pièces justificatives. De plus, nous devons ajouter comme fonctionnalité, la possibilité de pouvoir importer les absences d'un cours à partir d'un fichier (*csv, Json*).

Puis, nous devons aussi être en mesure de générer des listes d'absences de cours et de générer une liste des absences d'un étudiant et le total de ses absences justifiées et injustifiées.

Voici donc, toutes les **tâches / fonctionnalités**, à créer et à mettre en place, dans notre site web de gestion des absences. Ces différentes tâches et fonctionnalités, vont pouvoir être décrites dans la suite de ce document.

2) Diagramme de Gantt et Planning

Avant toute chose, il faut d'abord mettre en place ce que l'on appelle un : **Diagramme de Gantt**. Celui-ci va nous permettre de planifier, et de répartir les différentes tâches et activités à créer et à mettre en place. Du fait de sa chronologie, le Diagramme de Gantt va nous permettre de visualiser, et de suivre le déroulement du projet, et d'identifier les possibles retards ou problèmes, qui pourraient intervenir durant la création de notre projet. Voici donc, ci-dessous, le diagramme de Gantt de notre projet :

SAÉ2.03 METTRE EN PLACE UNE SOLUTION INFORMATIQUE POUR L'ENTREPRISE

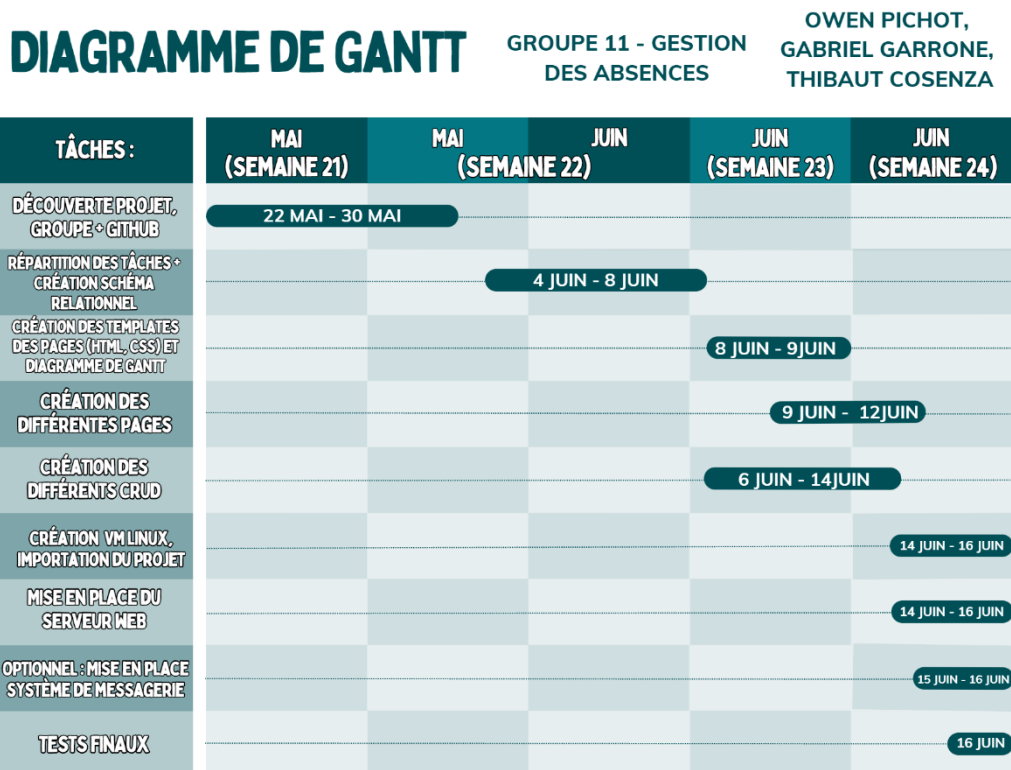


Figure 1 : Diagramme de Gantt de notre projet

Nous avons également pu, réaliser une **répartition des tâches**, qui a pu évoluer au cours du projet :

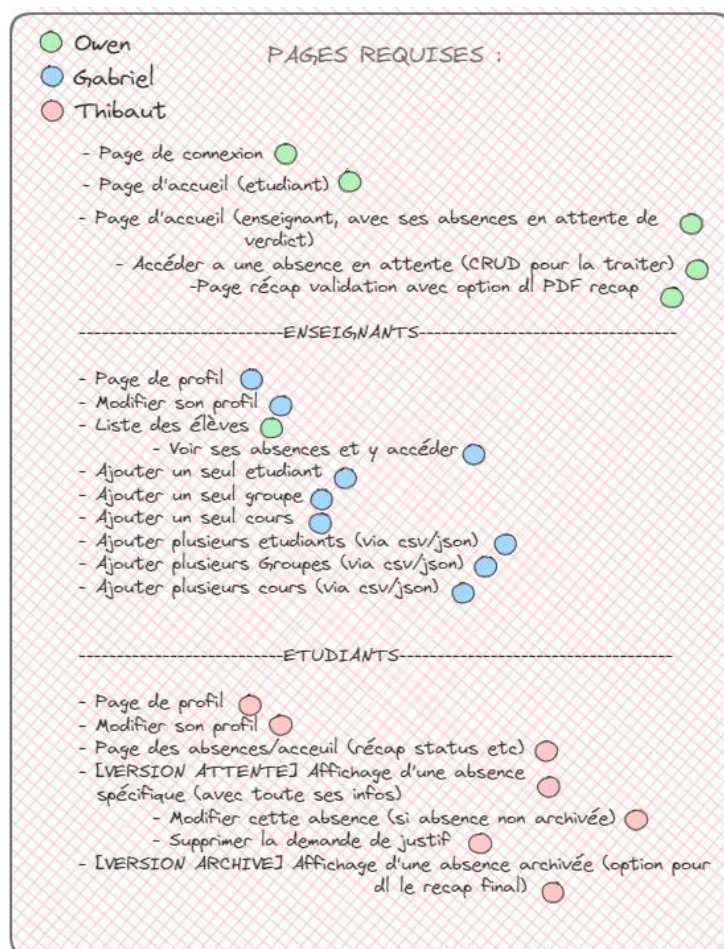


Figure 2 : Planning de notre projet (répartition des tâches)

3) Schéma relationnel du projet

Après avoir créer le diagramme de Gantt, et nous être répartis les tâches, nous avons donc créer un schéma relationnel pour notre projet de base de données. En effet, il fallait donc définir la structure et les relations entre les tables, pour que les données soient cohérentes entre elles, et pour garantir leur intégrité. Cela nous a aussi permis de représenter, les données et les différentes tables pour y voir plus clair.

Pour ce faire, nous avons pu créer un « **espace de travail collaboratif** » avec le logiciel en ligne nommé « **Excalidraw** ». Nous avons par exemple pu créer, le planning (vu précédemment), ou encore, faire le schéma relationnel du projet sur cette même page :

<https://excalidraw.com/#room=22fb39ca8e7b237d88e9,t2-EOKdqlGOkowkR7Qwt3A>

La table nommée « **Archives** », n'a finalement pas été implanté.

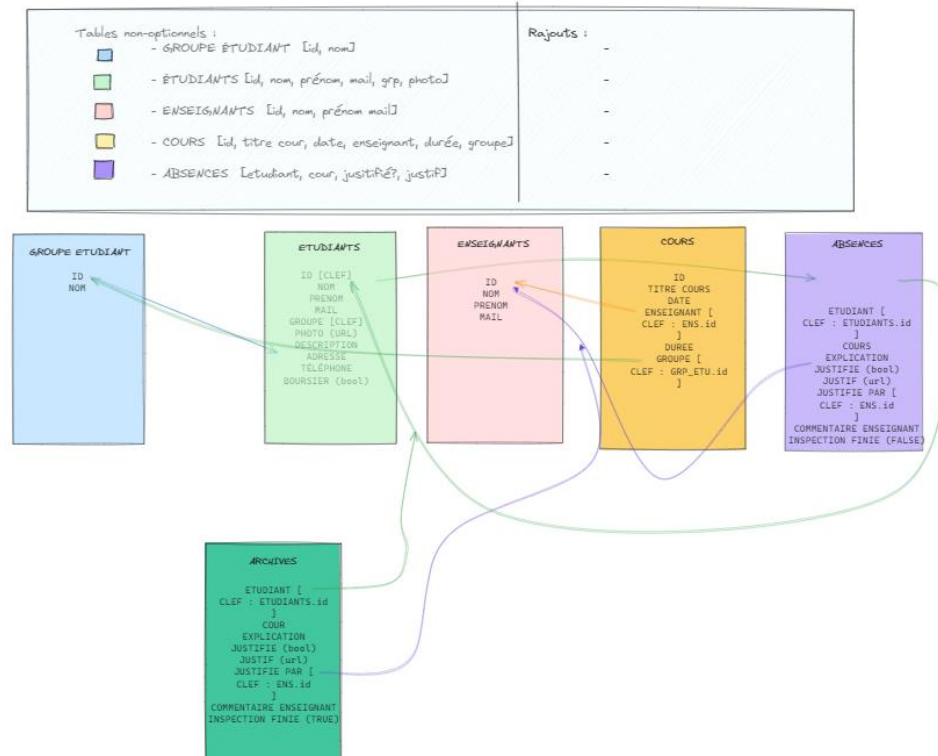


Figure 3 : Schéma relationnel de notre projet

Nous avons pu, refaire une version « améliorée » et plus claire, du schéma relationnel. Celle-ci est la version **finale** de notre schéma relationnel :

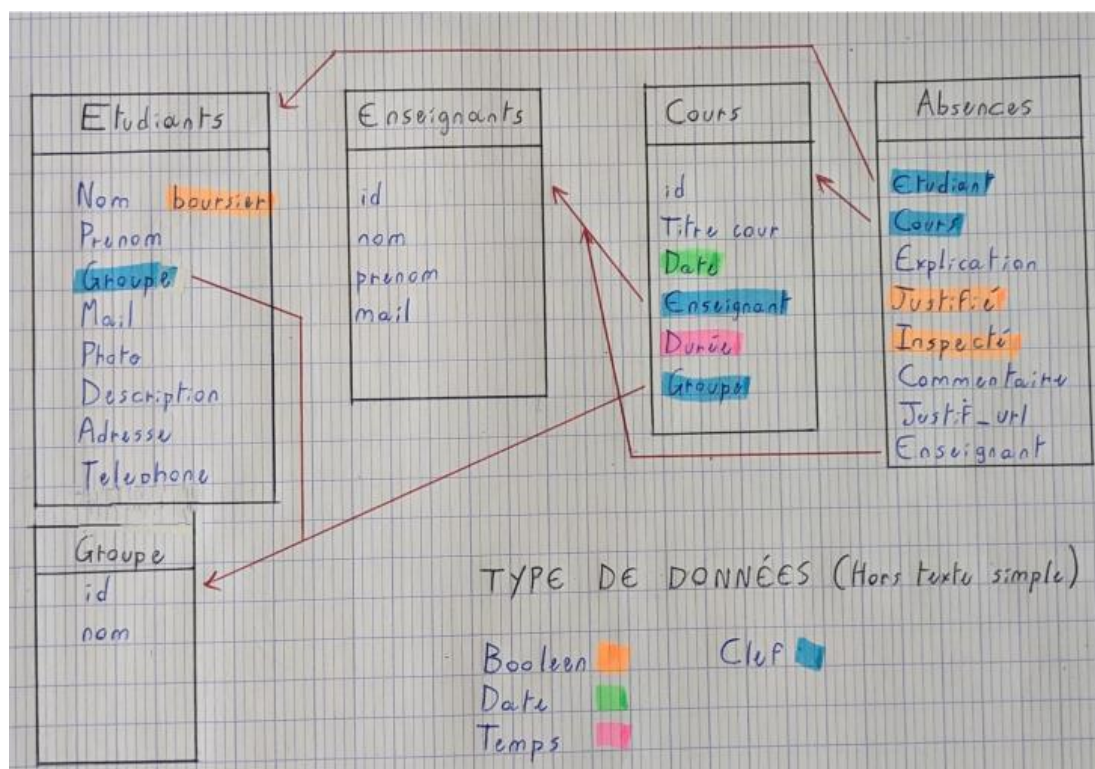


Figure 4 : Version finale de notre schéma relationnel

4) Création du projet Django pour notre projet

Pour commencer notre projet, il a donc fallu mettre en pratique, nos connaissances vues sur **Django** durant l'année. Il a donc fallu créer notre projet avec l'aide de **Django**. Nous avons pu également utiliser de l'HTML et du CSS, pour nos différentes pages. Pour faire tout cela, nous avons donc suivi les commandes suivantes :

⇒ *pip install django*

Installe le package **django** sur notre pc.

⇒ *django-admin startproject nom_du_projet*

Permet de créer notre **projet Django** (ici : *Absences*)

⇒ *python manage.py startapp nom_application*

Permet de créer notre **application**.

⇒ *'nom_app.apps.Nom_appConfig'*

Partie, à ajouter dans le fichier : **settings.py** dans la section *INSTALLED_APPS*.

⇒ *'python manage.py runserver'*

Permet de lancer le **serveur** avec Django.

⇒ *'python manage.py makemigrations'*

⇒ *'python manage.py migrate'*

Permet d'effectuer les **migrations** lorsque l'on a effectué une modification dans le fichier **models.py**.

Après avoir exécuté ces commandes, nous avons pu commencer à programmer et à coder pour la création de notre projet de **gestion des absences**.

Notre projet a été réalisé sur **Ubuntu** et sur **Windows** avec le logiciel : **Visual Studio Code**.

5) Base de données MySQL

Pour commencer le projet, il a donc bien fallu créer une base de données **MySQL** pour pouvoir créer les différentes tables de données pour notre projet (*citées plus haut*). **Owen PICHOT**, s'est occupé de créer la base de données et les tables au niveau de MySQL. Quelques **captures d'écrans** et **Screenshot**, vont pouvoir prouver cela.

En regardant la base de données, l'on s'aperçoit par exemple qu'il y a une base de données nommée « **DBABSENCES** » qui a pu être créée à partir du fichier **settings.py** de notre projet, avec différentes informations à l'intérieur, qui seront expliquées dans le prochain point de ce document :


```
mysql> show DATABASES;
+-----+
| Database |
+-----+
| DBASENCES |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0,00 sec)
```

Figure 5 : Tables

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'DBASENCES',
        'USER': 'thib',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'OPTIONS': {
            'autocommit': True,
        },
    },
}
```

Figure 6 : Partie du code dans le fichier (settings.py)

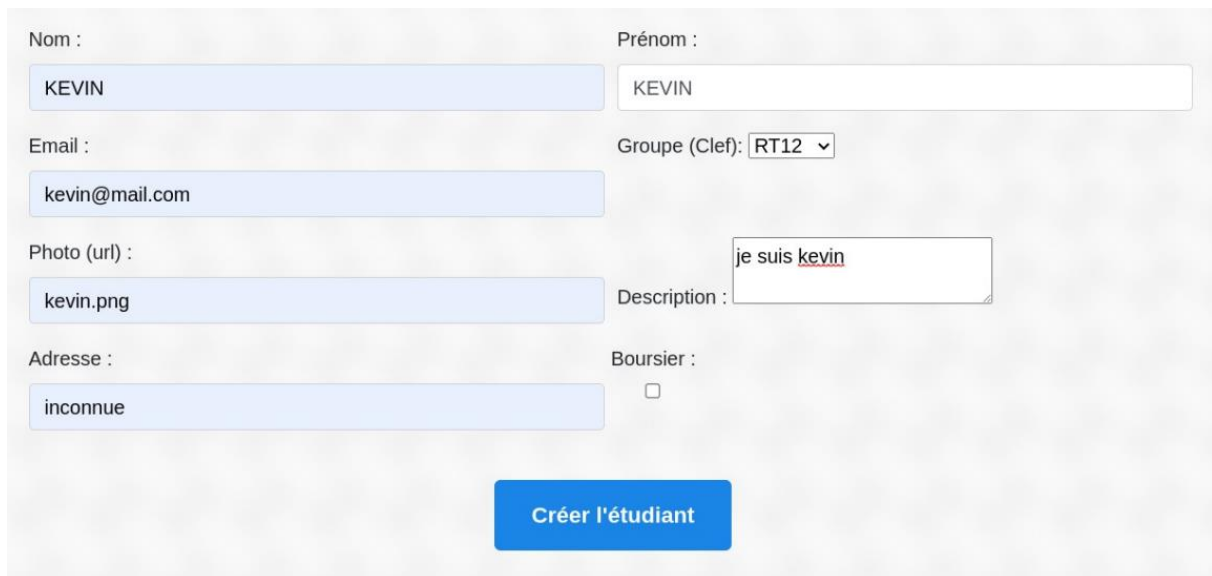
En tapant la commande : 'use DBASENCES', on peut voir toutes les tables créées :

```
mysql> use DBASENCES;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_DBASENCES |
+-----+
| ABSENCES_absences |
| ABSENCES_archives |
| ABSENCES_cours |
| ABSENCES_cours_Groupes |
| ABSENCES_enseignant |
| ABSENCES_etudiants |
| ABSENCES_groupeetu |
| auth_group |
| auth_group_permissions |
| auth_permission |
| auth_user |
| auth_user_groups |
| auth_user_user_permissions |
| django_admin_log |
| django_content_type |
| django_migrations |
| django_session |
+-----+
17 rows in set (0,00 sec)
```

Figure 7 : Liste des différentes tables créées à l'aide de MySQL

Après avoir fait les différents liens avec ces tables, et après avoir créer un **utilisateur** possédant des droits pour se connecter, on peut alors commencer à remplir un **premier** formulaire à partir de notre site web :



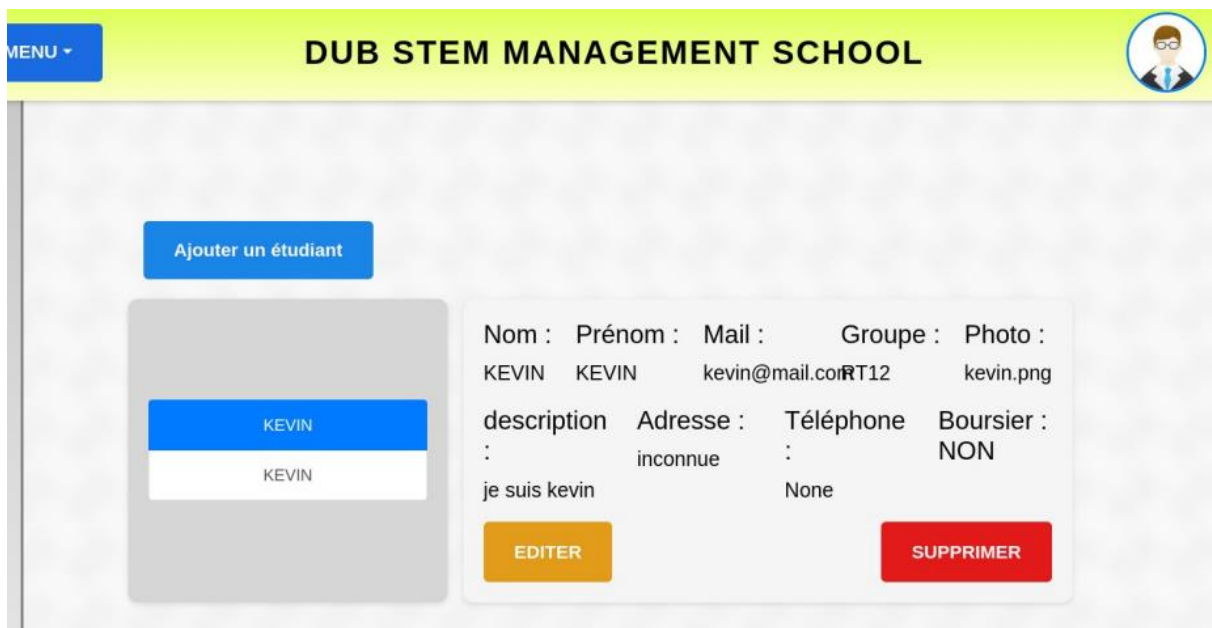
The form is titled 'Ajout d'un étudiant' and contains several input fields and a button. The fields are arranged in two columns. The first column contains 'Nom :', 'Email :', 'Photo (url) :', and 'Adresse :'. The second column contains 'Prénom :', 'Groupe (Clef):', 'Description :', and 'Boursier :'. The 'Groupe (Clef):' field is a dropdown menu with 'RT12' selected. The 'Boursier :' field is a checkbox. A blue button labeled 'Créer l'étudiant' is at the bottom center.

Nom :	KEVIN	Prénom :	KEVIN
Email :	kevin@mail.com	Groupe (Clef):	RT12
Photo (url) :	kevin.png	Description :	je suis kevin
Adresse :	inconnue	Boursier :	<input type="checkbox"/>

Créer l'étudiant

Figure 8 : Remplissage d'un premier formulaire (ici : Ajout d'un étudiant)

Celui-ci va alors bien apparaître sur la **page principale**, de l'ajout d'un étudiant :



The main page has a yellow header with 'MENU' and 'DUB STEM MANAGEMENT SCHOOL'. A blue button 'Ajouter un étudiant' is at the top left. Below it is a card for the student 'KEVIN'. The card displays the student's details in a table-like format. At the bottom of the card are two buttons: 'EDITER' (orange) and 'SUPPRIMER' (red).

Nom :	Prénom :	Mail :	Groupe :	Photo :
KEVIN	KEVIN	kevin@mail.com	RT12	kevin.png
description :	Adresse :	Téléphone :	Boursier :	
je suis kevin	inconnue	None	NON	

EDITER SUPPRIMER

Figure 9 : L'Étudiant ajouté apparaît bien dans la page principale

La Base de données est d'ailleurs disponible sur le lien **GitHub** de notre projet à l'adresse :

⇒ <https://github.com/Wewenito/SAE23.git>

6) Déploiement de notre projet

Après cela, il a fallu déployer notre site web. En effet, pour le déployer localement ou non, il faut alors se rendre dans le fichier **settings.py** pour **modifier différents paramètres** :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'DBABSENCES',
        'USER': 'thibaut',
        'PASSWORD': 'j38fqt',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'OPTIONS': {
            'autocommit': True,
        },
    },
}
```

Figure 11 : DATABASES modification dans settings.py

```
ALLOWED_HOSTS = [
    '127.0.0.1',
    '192.168.1.10', #this pc (ethernet)
    '192.168.1.1', #this pc (wi-fi)
    '176.153.132.17',
    '192.168.0.14', #adresse IP de mon PC
]
```

Figure 10 : ALLOWED_HOSTS modification dans settings.py

D'abord, dans la section « **DATABASES** », on retrouve la boucle locale ici « **localhost** » avec la redirection du port (**ici 3306**). Pour notre projet, c'était Owen qui hébergeait le site, sur son **serveur personnel**. Pour **héberger** sur son adresse personnelle, il a donc été nécessaire de modifier dans les paramètres de sa boxe, la redirection du port, qui pour notre projet, sera le **port 5**.

Puis, dans la section « **ALLOWED_HOSTS** » on retrouve les adresses possibles pour se connecter au site. On y voit donc par exemple, **l'adresse de bouclage (127.0.0.1) pour le local**, les **adresses d'Owen Pichot sur ces différents interfaces (Ethernet et Wifi)** et **l'adresse IP de mon PC**.

On accède au site web en **tapant cette adresse** :

176.153.132.17:5

On arrive donc sur la page principale pour se connecter en tant qu'**étudiant ou enseignant** :



Figure 12 : Page principale du site

Quelques captures d'écran des différentes fonctionnalités du site :

Ajout d'une absence chez l'étudiant :



Figure 13 : Ajout d'une absence (côté étudiant)

Fenêtre de **visualisation des absences** (acceptées, en attente, refusé) :

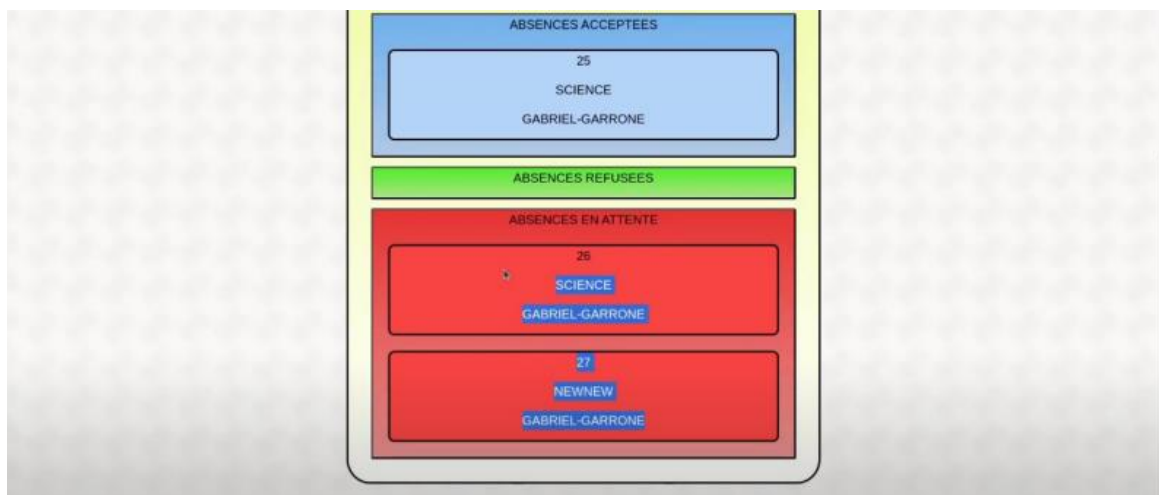


Figure 14 : Visualisation des absences (côté étudiant)

Visualisation des absences (côté enseignant) :

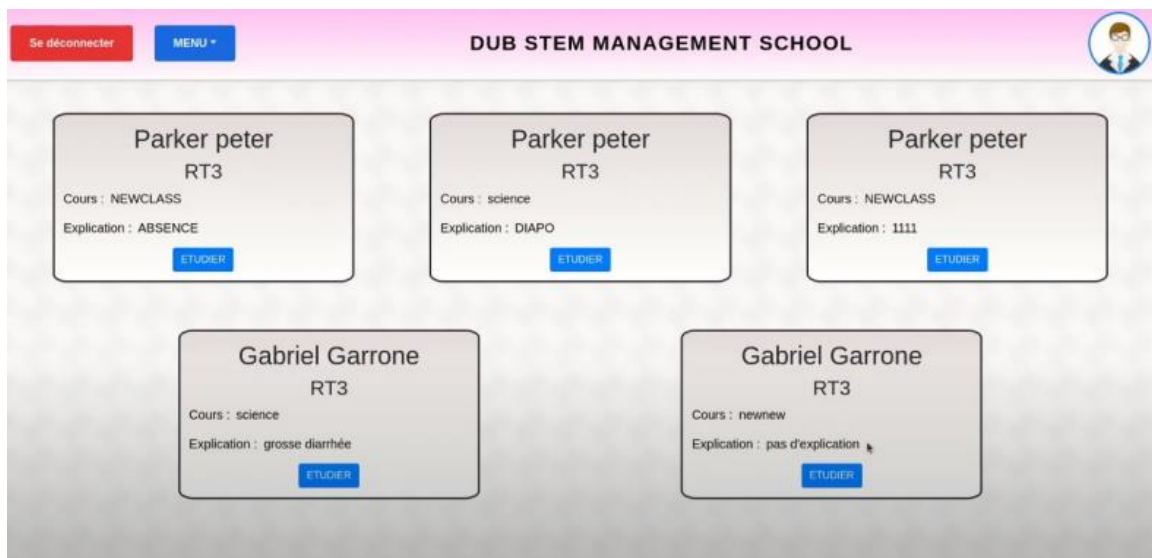


Figure 15 : Visualisation des absences (côté enseignant)

En cochant la case « **Justifie bool** » du côté **enseignant**, celui-ci accepte l'absence :

The screenshot shows a web form with the following fields and values:

- Etudiant: Gabriel-Garrone (dropdown menu)
- Cours: science (dropdown menu)
- Explication: grosse diarrhée (text input)
- Justifie bool: ☒ (checkbox)
- Justif url: Currently: justifications/hug.png (text)
- Change: Choisir un fichier (button) / Aucun fichier choisi (text)
- Checke par: John-CACA (dropdown menu)
- Commentaire enseignant: pas de comm (text input)
- Inspection terminee: ☒ (checkbox)
- VALIDER (yellow button)

Figure 16 : Acceptation de l'absence avec "Justifie bool"

A nouveau, sur le point de vue de l'étudiant, on remarque que **l'absence acceptée** précédemment par **l'enseignant** a été mise dans la case **Acceptée (chez l'étudiant)** :

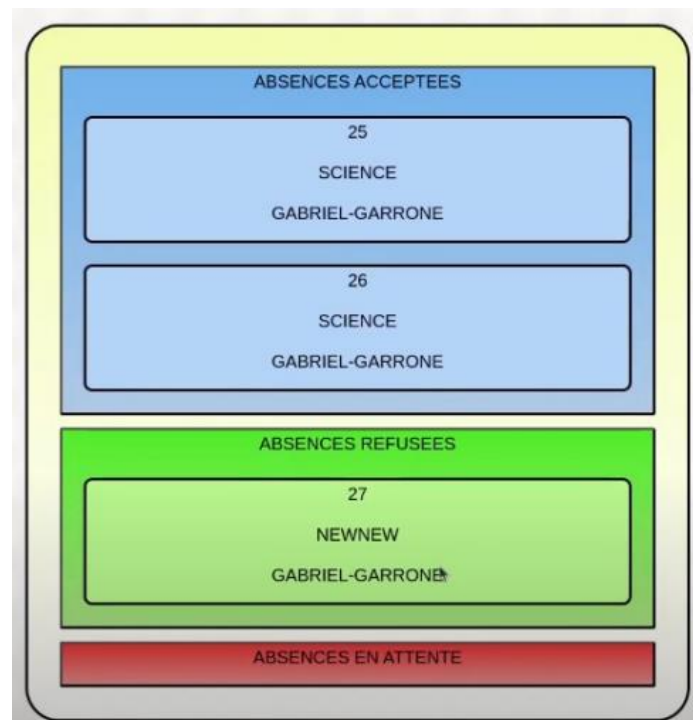


Figure 17 : Visualisation des absences chez l'étudiant (absence acceptée)

7) Fonctionnement du site

Pour que tout cela fonctionne, il faut impérativement créer différentes fonctions dans le fichier **views.py**, pour chaque fonctionnalité mise dans le projet. En voici **quelques exemples** :

```
def ajout_groupe(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            form = GroupeEtuForm(request.POST)

            if form.is_valid():
                form.save()
                return Afficher_groupes(request)
            else:
                form = GroupeEtuForm()
        return render(request, 'MAIN/groupes/Ajout_groupe.html', {'form': form})
    else:
        msg = 'Vous ne pouvez pas ajouter de groupe sans être connecté..'
        form = AuthenticationForm(request.POST)
        return render(request, 'registration/login.html', {'form': form, 'msg': msg})

def Afficher_enseignants(request):
    enseignant = Enseignant.objects.all()
    return render(request, 'MAIN/enseignants/Afficher_enseignants.html', {'enseignant': enseignant})
```

Figure 18 : Fonction : ajout d'un groupe et affichage enseignants

```
def Ajouter_enseignants(request):
    if request.user.is_authenticated:
        if request.method == 'POST':
            form = EnseignantForm(request.POST)
            if form.is_valid():
                enseignant = form.save(commit=False)
                enseignant.save()

                user = User.objects.create_user(username=enseignant.mail, password='toto')
                enseignant.user = user
                enseignant.save()

                enseignant_groupe = Group.objects.get(name='ENSEIGNANTS')
                user.groups.add(enseignant_groupe)

                return Afficher_enseignants(request)
            else:
                form = EnseignantForm()
        return render(request, 'MAIN/enseignants/Ajouter_enseignants.html', {'form': form})
    else:
        msg = 'Vous ne pouvez pas ajouter d\'enseignants sans être connecté..'
        form = AuthenticationForm(request.POST)
        return render(request, 'registration/login.html', {'form': form, 'msg': msg})

    return render(request, 'MAIN/enseignants/Ajouter_enseignants.html')

def Afficher_cours(request):
    cours = Cours.objects.all()
    return render(request, 'MAIN/cours/Afficher_Cours.html', {'cours': cours})
```

Figure 19 : Fonction : ajout d'un enseignant et afficher un cours

Il ne faut pas aussi oublier de créer les **différentes « routes »** dans le fichier **urls.py**, pour que l'utilisateur soit redirigé au bon endroit avec les différents chemins.

Lorsque l'on affiche une page qui correspond à un **ID** d'un groupe, d'un cours, il est nécessaire d'ajouter **<int :id>** pour que la page s'affiche suivant l'**ID** :

```
urlpatterns = [
    path('home/', views.home, name='home'),

    path('EasterEgg/', views.EasterEgg, name='EasterEgg'),

    path('signin/', views.signin, name='signin'),

    #/////////////////////////////////////////////////////////////////etudiant////////////////////////////////////

    path('gestion_etudiants/<str:arg1>', views.gestion_etudiants, name='gestion_etudiants'),
    path('accueil_etu/<int:id>', views.acueil_etu, name='accueil_etu'),
    path('Ajouter_absence/', views.Ajouter_absence, name='Ajouter_absence'),
    path('afficher_etude_absence/<int:id>', views.afficher_etude_absence, name='afficher_etude_absence'),
    path('etudier_absence/<int:id>', views.etudier_absence, name='etudier_absence'),

    #/////////////////////////////////////////////////////////////////enseignant////////////////////////////////////

    path('Afficher_etudiants/', views.Afficher_etudiants, name='Afficher_etudiants'),
    path('Ajouter_etudiants/', views.Ajouter_etudiants, name='Ajouter_etudiants'),
    path('supprimer_etudiant/<int:id>', views.supprimer_etudiant, name='supprimer_etudiant'),
    path('modifier_etudiant/<int:id>', views.modifier_etudiant, name='modifier_etudiant'),
    path('afficher_modif_form/<int:id>', views.afficher_modif_form, name='afficher_modif_form'),

    path('profil/update_mail_ens/<int:id>', views.update_mail_ens, name='update_mail_ens'),

    path('Gestion_enseignants/<str:arg1>', views.Gestion_enseignants, name='Gestion_enseignants'),

    path('Afficher_groupes/', views.Afficher_groupes, name='Afficher_groupes'),
    path('ajout_groupe/', views.ajout_groupe, name='ajout_groupe'),
    path('suppression_groupe/<int:id>', views.suppression_groupe, name='suppression_groupe'),
    path('modification_groupe/<int:id>', views.modification_groupe, name='modification_groupe'),

    path('Ajouter_cour/', views.Ajouter_cour, name='Ajouter_cour'),
    path('Afficher_cours/', views.Afficher_cours, name='Afficher_cours'),
    path('Supprimer_cour/<int:id>', views.Supprimer_cour, name='Supprimer_cour'),
    path('afficher_modif_form_cour/<int:id>', views.afficher_modif_form_cour, name='afficher_modif_form_cour'),
    path('modifier_cour/<int:id>', views.modifier_cour, name='modifier_cour'),

    path('Afficher_enseignants/', views.Afficher_enseignants, name='Afficher_enseignants'),
    path('Ajouter_enseignants/', views.Ajouter_enseignants, name='Ajouter_enseignants'),
    path('Supprimer_enseignant/<int:id>', views.Supprimer_enseignant, name='Supprimer_enseignant'),
    path('afficher_modif_form_ens/<int:id>', views.afficher_modif_form_ens, name='afficher_modif_form_ens'),
    path('modifier_enseignant/<int:id>', views.modifier_enseignant, name='modifier_enseignant'),
```

Figure 20 : Contenu du fichier urls.py

De plus, il est aussi intéressant voire même important de remplir les fichiers **forms.py** et **models.py** qui contiennent **respectivement** des classes de **formulaire** qui permettent de collecter, valider et traiter les données saisies par l'utilisateur, et des **modèles** de données pour gérer les relations entre les tables, valider les données et faciliter l'interaction avec la base de données. J'ai notamment pu m'occuper de cette partie, en créant les différents **CRUDS** (*de bases avant modification*) pour le projet.

```
from django import forms
from .models import GroupeEtu, Cours, Etudiants, Enseignant, Absences

class GroupeEtuForm(forms.ModelForm):
    class Meta:
        model = GroupeEtu
        fields = ['nom']

class CoursForm(forms.ModelForm):
    Groupes = forms.ModelChoiceField(queryset=GroupeEtu.objects.all())
    enseignant = forms.ModelChoiceField(queryset=Enseignant.objects.all())

    class Meta:
        model = Cours
        fields = [
            'titre_cours',
            'enseignant',
            'Groupes',
            'Date',
            'Duree',
        ]

class EtudiantsForm(forms.ModelForm):
    groupe = forms.ModelChoiceField(queryset=GroupeEtu.objects.all())

    class Meta:
        model = Etudiants
        fields = ['nom', 'prenom', 'mail', 'groupe', 'photo', 'description', 'adresse', 'telephone', 'boursier']

class EnseignantForm(forms.ModelForm):
    class Meta:
        model = Enseignant
        fields = ['nom', 'prenom', 'mail']

class AbsenceForm(forms.ModelForm):
    Checke_par = forms.ModelChoiceField(queryset=Enseignant.objects.all(), required=False)

    class Meta:
        model = Absences
        fields = ['Etudiant', 'Cours', 'Explication', 'Justifie_bool', 'Justif_url', 'Checke_par', 'Commentaire_enseignant', 'Inspection_terminee']
```

Figure 21 : Fichier forms.py

```
from django.db import models

# Create your models here.

class Enseignant(models.Model):
    nom = models.CharField(max_length=50, blank=False, null=False)
    prenom = models.CharField(max_length=50, blank=False, null=False)
    mail = models.CharField(max_length=50, blank=False, null=False)

    def __str__(self):
        return f"{self.nom}-{self.prenom}"

class GroupeEtu(models.Model):
    nom = models.CharField(max_length=50, blank=False, null=False)

    def __str__(self):
        return f"{self.nom}"

class Cours(models.Model):
    Titre_cours = models.CharField(max_length=50, blank=False, null=False)
    enseignant = models.ForeignKey("Enseignant", on_delete=models.CASCADE, default=None, blank=True, null=True)
    Groupes = models.ForeignKey("GroupeEtu", on_delete=models.DO_NOTHING, default=None)
    Date = models.DateTimeField(blank=True, null=True)
    Duree = models.TimeField(blank=True, null=True)

    def __str__(self):
        return f"{self.Titre_cours}"

class Etudiants(models.Model):
    nom = models.CharField(max_length=50, blank=False, null=False)
    prenom = models.CharField(max_length=50, blank=False, null=False)
    mail = models.CharField(max_length=50, blank=False, null=False)
    groupe = models.ForeignKey("GroupeEtu", on_delete=models.CASCADE, default=None)
    photo = models.CharField(max_length=100, blank=True, null=True)
    description = models.CharField(max_length=300, blank=True, null=True)
    adresse = models.CharField(max_length=100, blank=True, null=True)
    telephone = models.IntegerField(blank=True, null=True)
    boursier = models.BooleanField()
```

Figure 22: Fichier models.py

8) Apparence du site avec HTML et CSS

Pour la partie **esthétique** du site, c'est **Gabriel GARRONE** qui s'en est chargé. En effet, nous voulions en termes d'esthétique du site, une plateforme moderne. C'est pourquoi, Gabriel a pu s'occuper d'une bonne partie du développement des pages et de l'esthétique de celles-ci. Voici la liste des **différentes pages HTML et CSS** créées :

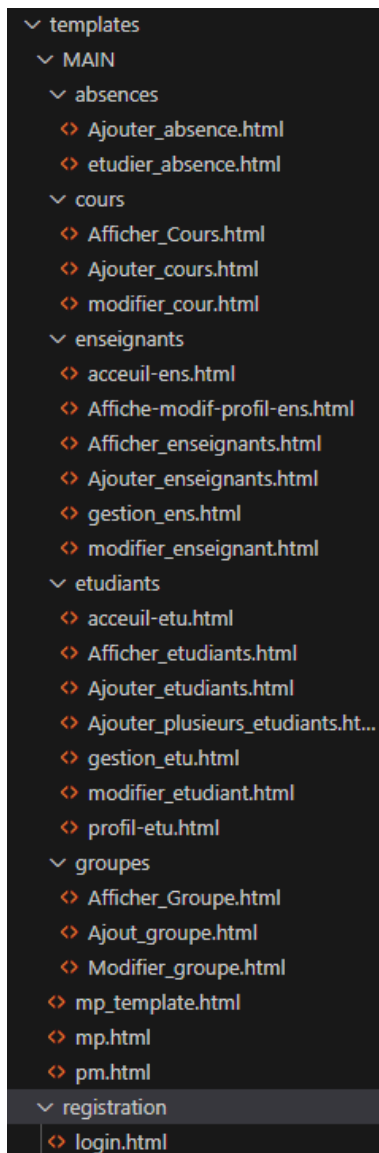


Figure 24 : Liste des pages HTML créées

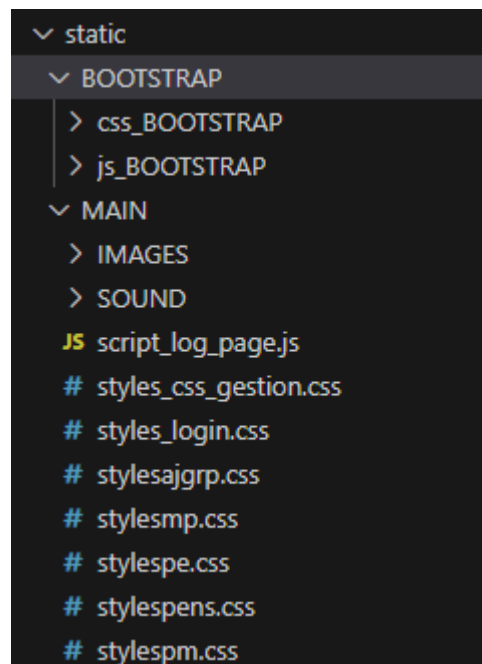


Figure 23 : Liste des pages CSS créées

Nous avons d'ailleurs mis en place un **squelette** pour le **header**.

La fonction « **load static** » va permettre de charger nos fichiers statiques (styles, images...) :

```
{% extends 'MAIN/mp_template.html' %}

{% load static %}

{% block HEAD %}
{% endblock %}
```

Figure 25: Fonction "load static"

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    {% load static %}

    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="{% static 'BOOTSTRAP/js_BOOTSTRAP/bootstrap.min.js' %}"></script>
    <link rel="stylesheet" type="text/css" href="{% static 'BOOTSTRAP/css_BOOTSTRAP/bootstrap.min.css' %}" >
    <link rel="stylesheet" type="text/css" href="{% static 'MAIN/stylesheet.css' %}">
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"></script>

    {% block HEAD %}
    {% endblock %}

    <title>GESTION ABSENCES [HOME]</title>
  </head>
  <body>

    <header class="navbar navbar-dark bg-dark id='headermainp'">
      {% block HEADER %}
        {% if user.is_authenticated %}
          {% for group in user.groups.all %}
            {% if group.name == 'ETUDIANTS' %}
              <div id="left_header_div">
                <a href="{% url 'logout' %}"><button type="button" id="MENU_BUTTON">Se déconnecter</button></a>
                <div id="dropdownlinks" class="LOGOUT_BUTTON">
                  <a class="dropdown-toggle" href="#" role="button" id="dropdownMenuLink" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                    MENU</a>
                  <div class="dropdown">

                    <div class="dropdown-menu" aria-labelledby="dropdownMenuLink">
                      <a class="dropdown-item" href="{% url 'ABSENCES:accueil_etu' user.id %}">GESTION ABSENCES</a>
                      <hr>
                      <a class="dropdown-item" href="{% url 'ABSENCES:home' %}">PAGE PRINCIPALE</a>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          <div id="title_header_div">
            <span class="titre"> Dub STEM Management School </span>
          </div>
        </div>
      </div>
    </body>
```

Figure 26 : Fichier mp_template.html

9) Conclusion du projet / SAÉ

Ce projet de développement d'un site web de gestion d'absences a été une expérience **assez bien mené en soi**. Cela m'a permis notamment de me familiariser avec l'univers de Django avec les différents fichiers à configurer / paramétrer.

Seulement, j'ai pu avoir beaucoup de mal à comprendre certaines choses dans cette SAÉ. Notamment, comment relier une base de données MySQL, ou encore comment relier les différents CRUDS entre eux. Heureusement, j'ai pu être aidé et épaulé par mes deux collègues **Owen PICHOT** et **Gabriel GARRONE**, qui ont su répondre à mes questions, et régler mes problèmes de compréhension sur certains sujets ou encore sur des problèmes **d'erreurs** du logiciel.

En termes de compétence, travailler sur ce projet m'a permis de gagner en compétence sur ce qui est du domaine de la gestion de projet, avec la création du diagramme de Gantt par exemple ou encore planifier et organiser notre temps et nos ressources pour le projet.

Un **point à noter**, est que j'ai ressenti qu'il manquait du temps pour finaliser ce projet. En effet, j'ai trouvé que ce projet manquait de temps de travail en plus pour pouvoir être complètement achevé et terminé.

Dans l'ensemble, ce projet m'a quand même appris beaucoup de choses, notamment sur le fonctionnement de Django, ou encore le fonctionnement d'un site web.