Dossier-projet Informatique et Sciences du Numérique

« Petit Laboratoire Numérique »

Sommaire

Présentation générale du projet......................................................................page 2 - 3

Explications et commentaires sur ma tâche personnelle dans le projet.........pages 3 - 9

Rendu final du projet......................................................................................page 10

1

Partenaires

Jérémie Blanchart

Romane Vallée

Présentation du projet

Nous avions pour projet la création d'un site Web utilitaire pour la Physique-Chimie qui permettrait notamment à l'utilisateur, en rentrant simplement des données comme « Concentration Molaire de la solution mère », « Concentration Molaire de la solution fille » et « Volume de solution fille voulu » de se voir retourner le volume de solution mère à prélever. Ou encore en rentrant « concentration molaire de la solution », « volume de solution », « Masse molaire de l'espèce chimique » de se voir retourner la masse à peser pour concevoir la solution.

Pour réaliser ce projet, il nous a fallu réaliser une partie de codage HTML et CSS pour la page, ainsi que plusieurs fonctions python pour la calculatrice (qui se sont retrouvées dans le codage du serveur). Il nous a donc aussi fallu élaborer un mini-serveur capable de traiter les données saisies par l'utilisateur pour retourner la réponse adéquate.

Ma motivation personnelle pour ce projet

Ce qui me plaisait particulièrement dans ce projet était la perspective d'aller plus loin dans le codage de sites Internet, ayant déjà apprécié le devoir maison de préparer une page Web pendant l'année. Avec l'ajout d'un serveur capable de traiter des données rentrées par l'utilisateur, cela représentait pour moi un approfondissement de ce qui avait été appris pendant l'année. De plus, ce projet se distinguait de la majorité des autres projets des autres groupes, qui étaient principalement des jeux.

Répartition des tâches

Nous avons divisé le travail en trois parties que nous nous sommes attribués de la façon suivante :

Romane : codage HTML de la page

Jérémie : codage CSS, mise en forme des pages

Nicolas : conception du mini-serveur qui traite les données saisies par l'utilisateur (en python)

Compte-tenu des inter-dépendances entre les fichiers (serveur-html, serveur-sorties, html-css), une telle répartition des tâches induisait la nécessité d'une coopération continue entre les membres du groupe.

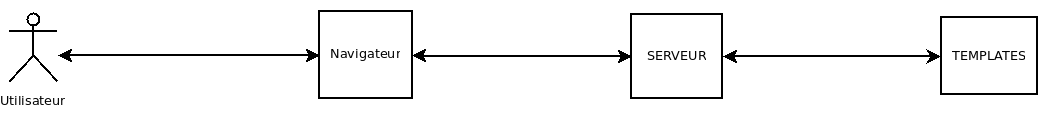
2

Nous avons utilisé une plateforme de mutualisation (Github) pour pouvoir travailler chez nous, en local, y mettre nos modifications de codages, de fichiers en ligne, et par la commande « git pull » que chacun puisse actualiser son contenu par rapport à celui en ligne.

Ma tâche

Ma tâche était donc de concevoir le codage du serveur qui traite les données rentrées par l'utilisateur. Je vais donc détailler mon travail dans ce qui suit. Commençons par expliquer le fonctionnement d'un serveur de manière générale :

Schéma du fonctionnement d'un serveur



La première étape, c'est à dire de l'utilisateur au navigateur, correspond au moment où l'utilisateur tape l'adresse du serveur dans la barre d'adresse de son navigateur.

Lorsque l'utilisateur appuie sur la touche « Entrée » commence l'étape entre le navigateur et le serveur. Le navigateur envoie une requête au serveur, qui va ensuite la traiter.

Ensuite, selon la requête, le serveur va exploiter un fichier appelé template (un fichier ici html sur lequel on se base pour la conception d'une page. Dans d'autres cas, il peut désigner un fichier de base pour la conception d'un logiciel) présent sur la machine hôte du serveur (dans notre cas par défaut le fichier « formulaire.html » voir suite) et renvoyer au navigateur le contenu de ce fichier, qui sera donc perçu par l'utilisateur.

3

Passons maintenant au codage du serveur en lui même :



Il est composé de différents blocs de codage remplissant chacun une fonction précise dans le fonctionnement du site.

Pour coder, j'ai fait appel à différentes ressources de la bibliothèque logicielle bottle (une bibliothèque étant une collection de ressources prêtes à être utiliser par des programmes).

4

L'ensemble des ressources que j'ai mobilisées sont résumées dans cette ligne :



resquest me permet de récupérer les données rentrées dans les formulaires

route me permet d'indiquer au serveur le chemin à suivre pour récupérer et traiter les données de tel ou tel formulaire.

run me permet de lancer le serveur

template me permet de retourner des fichiers templates évoqués précédemment au navigateur.

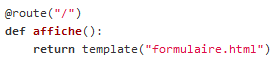
get permet au serveur de traiter une quantité assez limitée de données mais sans avertissement si on tente de rafraîchir. En d'autres termes, si un utilisateur nerveux appuie trois fois sur le bouton « Envoyer » par manque de patience, le serveur effectuera trois fois le calcul, mais sans redemander à chaque fois confirmation du renvoi des données.

J'aurais pu mobiliser post mais cela aurait été moins adapté. De fait, mon serveur ne traite que les données de 3 variables (voir suite) et un renvoi de ces données n'aurait pas de conséquence majeure. L'avantage de post étant de pouvoir traiter une quantité beaucoup plus importante de données, mais en demandant confirmation du renvoi des données en cas de tentative de rafraîchissement. C'est d'autre part pour cela que post est utilisé dans les transactions en ligne.

Enfin, static\_file permet au serveur de retourner des fichiers statiques (des fichiers dont le contenu n'est pas modifié pour une requête par opposition aux fichiers dynamiques qui génèrent un contenu en réponse à une requête) tel que les images ou les fichiers css. Sans, je ne pourrais afficher avec mon serveur que du contenu brut.

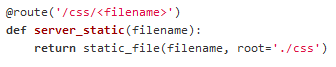
5

Passons maintenant au premier bloc de code :



Le @route(« / ») renvoie au chemin par défaut. Je définis ensuite la fonction affiche qui retourne le fichier template « formulaire.html ». Il s'agit en résumé de la page d'accueil lorsque l'on sollicite le serveur à l'adresse de base.

Le second bloc :



Il permet au serveur d'exploiter les fichiers css présent dans le dossier css, d'où le @route('/css/<filename>') et le root='./css' grâce à static\_file.



Le troisième :

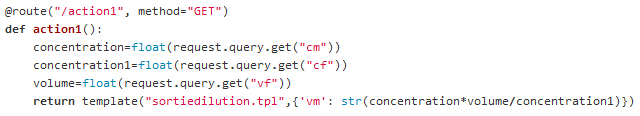


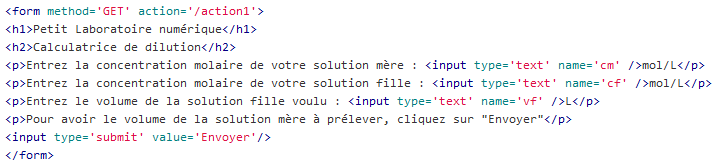
Même principe que pour le second, sauf qu'il s'agit des images présentes dans le dossier img



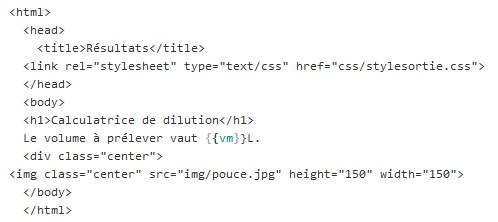


6

Le quatrième :

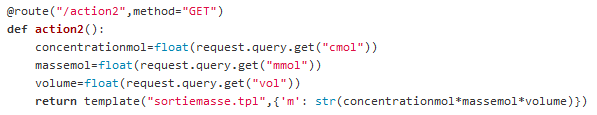


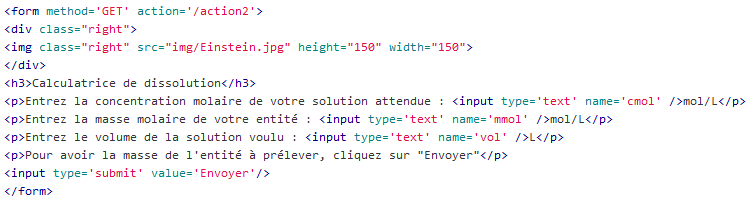
Ici, je ne pouvais pas mettre autre chose que @route(« action1 », method=« GET ») car dans le fichier formulaire.html, il y a <form method='GET' action='http://0.0.0.0:8000/action1'>. De même que je ne peux changer le nom des variables dont je récupère le contenu (cm, cf et vf) grâce à request et get. Ce contenu, je le transforme en float, et je le stocke dans d'autres variables (concentration, concentration1 et volume) pour pouvoir les utiliser dans l'opération qui suit. Je demande au serveur de renvoyer le résultat du calcul concentration\*volume/concentration1 dont je stocke le résultat dans une nouvelle variable vm dans le fichier sortiedilution.tpl.

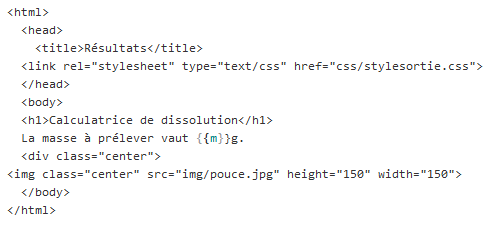


7

Le cinquième :







Le principe est le même que pour le quatrième bloc.

8

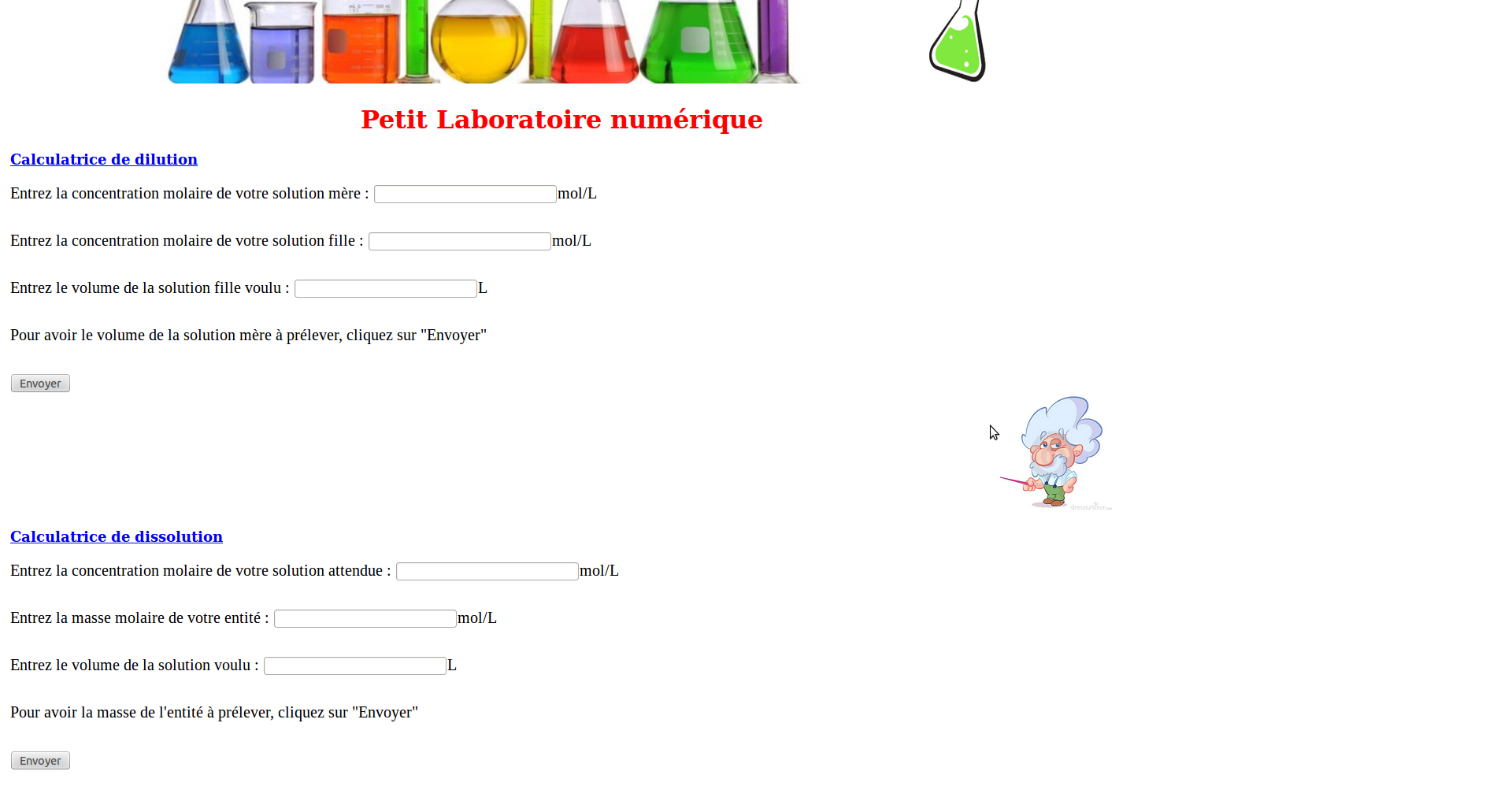
Et enfin, le sixième et dernier bloc, qui concerne le lancement du serveur :



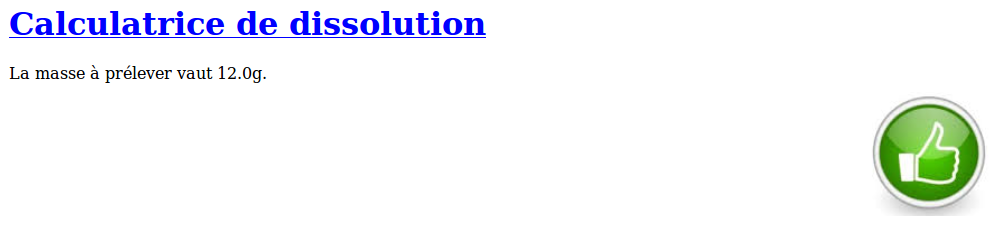
C'est là que run trouve son utilité. host=« 0.0.0.0 » renvoie à n'importe quelle ip. Comme les fichiers du projet sont sur une clé, suivant la machine auquel je la branche, l'ip change, d'où le renvoi à n'importe quelle ip, qu'il me suffit de trouver (l'ip de la machine hôte) pour savoir quoi taper dans la barre d'adresse du navigateur de la machine modélisant l'utilisateur. J'ai choisi de placer le serveur sur le port 8000 (port=8000). La fonction reloader que j'ai autorisé a permis au serveur de redémarrer à chaque modification du code que j'opérais.

9

Rendu final du projet







10