Campus d'Aix-en-Provence
Année 2021-2022

UEI-Organiser-Programmation web - Projet
mathias.kleiner@ensam.eu
arnaud.polette@ensam.eu
lucas.vergez@ensam.eu

Projet programmation web : application web pour Agi[lean|log]

Objectif

Le but du projet est de réaliser une application web permettant de gérer une partie de l'interface Agilean/Agilog/Agipart : la gestion des stocks d'AgiLog ainsi que les commandes de pièces et de kits. Il est proposé de réaliser cette application en se basant sur le micro-framework Flask (langage Python).
Vous vous diviserez initialement en 3 groupes de 8 étudiants.

Description et cahier des charges

Les différents documents disponibles sur SAVOIR, ainsi que votre expérience pratique sont votre principale source d'information sur le contexte. Le programme **final** doit permettre les fonctionnalités suivantes (à étudier et éventuellement modifier ou étendre) :

- (Agilean) Créer une commande de kits
- (Agilog) Créer une commande de pièces, voir les commandes de kits/pièces en cours, valider la réception de pièces, notifier de la fin de la préparation de kits, afficher les stocks, etc.
- (Agipart/Agigreen) Voir les commandes de pièces, notifier l'envoi d'une comande

La création/modification des kits/pièces disponibles (uniquement) pourra être effectuée directement via un outil de gestion/administration de bases de données.

Déroulement des séances et travail autonome

Séance 1 : répartition des rôles, étude du contexte, prise en main des outils

- discuter du projet et définir les fonctionnalités de manière plus précise : liste/organigramme des pages, schémas, workflows, etc.
- définir les sous-groupes correspondants aux différentes tâches :
  - chef de projet (1 personne) : planning, suivi, interfaces entre groupes, intégration et mise en commun
  - experts métiers (1-2 personnes) : définition fonctionnalités, conception et implémentation de la base de données, tests et retours critiques
  - designers HTML(/CSS) (1-2 personnes) : design et implémentation des pages/formulaires HTML/CSS puis des templates HTML pour Flask
  - programmeurs (3-4 personnes) : conception et réalisation de l'application Python/Flask (routes, vues, etc.)
- se familiariser avec les technologies et outils, lire la documentation, faire des essais (voir section technologies et outils correspondante à votre sous-groupe dans ce document)

Travail autonome avant séance 2 : Implémentations indépendantes

- implémentation BDD SQLite, lister les requêtes SQL (experts métiers)
- conception des pages/formulaires avec données factices (HTML/CSS)
- définition et implémentation des routes/vues sans templates HTML ni BDD (Python/Flask)
- étudier et comprendre interfaces entre sous-groupes (templates HTML, requêtes BDD)

Séance 2 : Mise en commun, interfaces entre sous-groupes

- interface entre HTML et Flask : adapter les pages sous formes de templates HTML, les utiliser dans l'application web
- interface entre Flask et la BDD : adapter les fonctions route/vue pour faire le lien avec la BDD (écriture et appel de fonctions effectuant les requêtes SQL)

Travail autonome avant séance 3 : Implémentations intégrables

- terminer l'implémentation des pages utilisant des templates paramétrés (HTML + Flask)
- terminer l'implémentation des fonctions de l'application liées à la BDD (Flask + BDD)
- tests de l'application (experts métiers)
- rapport et présentation (Chef de projet)

Séance 3 : intégration finale, livrables

- livrer l'application fonctionnelle ("livrable")
- présentation du travail effectué (slides)

Technologies et outils

Les informations générales sur les outils et leurs interactions sont présentes dans le cours (disponible sur SAVOIR). Une fois les sous-groupes crées, référez-vous à la section correspondante ci-dessous pour plus de détails sur votre partie.

(Chef de projet) Système de gestion de versions : GIT
Un système de gestion de versions est fortement recommandé lors du développement en groupe d'une application. Celui-ci permet d'avoir un dépôt centralisé des données où chacun peut apporter ses modifications, voir l'évolution du projet, créer des rapports de bug, gérer le cahier des charges, etc. Nous proposons d'utiliser GIT, et le service d'hébergement GIT populaire gratuit Github[1]. Le chef de projet doit, lors de la première séance, créer un dépot pour son groupe, se familiariser avec l'outil et, avant la deuxième séance, s'assurer que ses camarades comprennent et utilisent le dépôt pour leurs développements. Ceci peut se faire directement depuis certains outils de développement (comme Geany), ou en utilisant l'application web de github.

(Experts métiers) BDD : SQLite
Nous utiliserons un fichier plat SQLite comme base de données de l'application web.
Pour créer le fichier, la structure de la base de données et modifier les données, vous pouvez utiliser le logiciel (multi-plateforme, sous licence libre GPL3) "DB Browser for SQLite". Pour cela téléchargez et installez la version "portable" sur *http ://sqlitebrowser.org*
Une fois la structure actée et quelques données-exemple entrées, écrivez et testez les requêtes SQL qui vous paraissent nécessaires pour l'application web : sélection de lignes (ex. : liste des commandes en cours), insertion (ex. : ajout d'une nouvelle commande), etc.
Dans un deuxième temps, il vous faudra intégrer votre BDD à l'application Flask. Il vous faudra écrire la(les) requête(s) SQL correspondant à une fonctionnalité demandée par les développeurs. Des exemples simples sont disponibles sur SAVOIR (voir Flask ci-dessous).

(Designers) HTML/CSS
Vous pouvez créer les fichiers HTML/CSS avec un éditeur de texte comme Geany. Pour tester le résultat, il suffit d'ouvrir le fichier avec un navigateur web.
Référence des langages HTML/CSS : *https ://www.w3schools.com/html/*
Les navigateurs possèdent généralement des outils pour analyser les problèmes dans les pages HTML/CSS. Par exemple, sous Firefox, vous pouvez utiliser la barre "Outils de développement".
Une fois assimilées les bases de l'HTML/CSS, commencez à créer les squelettes des pages nécessaires à votre application web AGI : exemples de formulaires (ex. : nouvelle commande), exemples d'affichage des résultats (ex. : liste des commandes en cours), pages de navigation dans l'application, etc. A ce stade, vous ne disposez pas des informations de la BDD : utilisez des données factices statiques.
Dans un deuxième temps, il vous faudra transformer vos squelettes en templates HTML selon la syntaxe de Flask (Ninja). Cela consiste à remplacer les données factices par celles (dynamiques) qui seront transmises par l'application Python/Flask. Cela nécessite une coordination entre les deux sous-groupes. Commencez par comprendre le système de templates. Des exemples simples sont disponibles sur SAVOIR (voir ci-dessous).

1. https ://github.com

(Developpeurs) Flask (micro-framework Python pour le web)

Documentation pour Flask : *http ://flask.pocoo.org*

Nous utiliserons la librairie Flask pour implémenter l'application web ainsi que pour son serveur web local de développement.

Pour lancer un serveur web local de développement, il suffit d'inclure les instructions suivantes dans votre application web/python :

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  # placer les fonctions route/vue de l'application Flask ici
6  @app.route('/')
7  def index():
8
9      return 'OK !'
10
11 if __name__ == '__main__':
12   app.run(debug=True)
```

Lors de l'exécution de cette application, un serveur web local est crée. Les routes (= URLs) disponibles sont définies par vos fonctions route/vue. Le serveur web surveille les modifications du fichier et se relancera à chaque fois automatiquement. Si une erreur de compilaton survient à ce moment, le serveur s'arrête. Il faut alors corriger les erreurs et le relancer.

Pour vous familiariser avec Flask, récupérez l'archive "Flask - exemple" sur SAVOIR et décompressez-là. Ouvrez le fichier "exemples.py" dans votre éditeur Python (Geany), executez le programme et testez le résultat avec un navigateur web. Naviguez dans les différentes pages et regardez pour chacune comment elles sont implémentées dans le programme Python/Flask.

Une fois que vous avez testé, compris et assimilé les différentes possibilités de Flask démontrées dans cet exemple, essayez de construire vos propres fonctions route/vue, par exemple :

- une page qui affiche la valeur de deux paramètres GET passés dans l'URL
- une page qui affiche un formulaire demandant une valeur numérique et affiche cette valeur une fois le formulaire soumis
- une page qui affiche la liste des personnes de la BDD dont le prénom commence par C
- une page qui demande à l'utilisateur d'entrer une lettre et affiche la liste des personnes de la BDD dont le prénom commence par cette lettre
- une page qui affiche une page HTML écrite par un de vos camarades à laquelle vous ajoutez un paramètre de template
- etc.

Une fois que vous avez réalisé quelques essais, commencez à lister les différentes pages de votre application web AGI. Ecrivez les fonctions route/vue correspondantes avec les informations dont vous disposez. Dans un premier temps, utilisez des affichages simples (sans templates HTML), et des données factices (entrées directement dans le code sans utilisation de la BDD).
Au fur et à mesure des séances et de l'avancée des différents sous-groupes, vous adapterez ces fonctions avec l'utilisation de templates HTML et/ou les requêtes à la BDD.