

Développement web côté serveur : Python et HTML

Tout ce dont vous avez besoin pour ce module, dans un seul document. Cochez au fur et à mesure.

U6-S2-M5

module

Élève

document

TechnoVert

scénario 2

D · 5 h 30

format et durée

Ce qu'il faut savoir avant de commencer	4
C'est quoi une fonction en Python ?	4
C'est quoi un dictionnaire ?	4
Comment lire un message d'erreur Python ?	4
Comment joindre un programme sur le réseau ?	5
Quelques outils Python du labo	5
Vérifiez-vous	6
Ce que fait le serveur, ce que fait le navigateur	7
Capsule 1 sur 5 Qui fait quoi : le serveur ou le navigateur	7
Capsule 2 sur 5 Route, vue, gabarit : le chemin d'une page	9
Capsule 3 sur 5 Recevoir une saisie et la vérifier	11
Capsule 4 sur 5 Se souvenir de l'utilisateur : la session	13
Capsule 5 sur 5 Quand le serveur plante	15
Ce qu'on retient	16
Ce que fait le serveur, ce que fait le navigateur	17
L'essentiel	17
Les mots à connaître	17
La méthode : écrire une vue qui reçoit un formulaire	18
Un exemple concret	18
Les pièges	18
Un mini-site à trois pages	20
PARTIE A — Afficher la liste	21
PARTIE B — Recevoir un formulaire	25
PARTIE C — Vérifier et protéger	30
PARTIE D — Lire une erreur serveur	36
POUR ALLER PLUS LOIN	39
CE QU'IL FAUT RETENIR	40
L'interface de consultation des chantiers	41
CE QUE VOUS RENDEZ	41
PARTIE A — La page de liste	44

PARTIE B — Le filtre par service	45
PARTIE C — La page de détail	47
PARTIE D — La connexion et la lisibilité	48
Tous les documents du module	50
A1 — Le mini-site de départ et son mode d'emploi	50
A2 — Les gabarits du mini-site	53
A3 — Les données de TechnoVert et les comptes de test	58
A4 — Extrait de la documentation de Flask	61
A5 — Le point de départ du TP	62
A6 — Les écrans attendus	64
A7 — La charte de lisibilité minimale	66

Ce qu'il faut savoir avant de commencer

Ce qu'il faut savoir avant de commencer. Lisez-le si vous avez un doute. Environ 10 minutes.

C'est quoi une fonction en Python ?

Une fonction est un bloc de code qui porte un nom. On l'écrit une fois, on l'appelle quand on veut.

Elle reçoit des valeurs entre parenthèses et renvoie un résultat avec `return`.

```
def double(nombre):  
    return nombre * 2  
  
double(4)      # renvoie 8
```

C'est quoi un dictionnaire ?

Un dictionnaire range des valeurs sous des noms, appelés clés. On lit une valeur avec sa clé entre crochets.

```
chantier = {"id": 3, "nom": "Éclairage des allées", "statut": "ouvert"}  
chantier["nom"]      # "Éclairage des allées"  
"statut" in chantier  # True : la clé existe
```

Une liste de dictionnaires, c'est un tableau : une ligne par dictionnaire.

Comment lire un message d'erreur Python ?

Quand un programme plante, Python affiche une **trace**. On la lit **par le bas**.

La dernière ligne donne le type d'erreur et sa cause. Juste au-dessus, on trouve le fichier et le numéro de la ligne fautive.

```
File "calcul.py", line 4, in moyenne
    return total / nombre
ZeroDivisionError: division by zero
```

Ici, la ligne 4 de `calcul.py` divise par zéro : la variable `nombre` vaut 0.

Comment joindre un programme sur le réseau ?

Une machine a une adresse IP. Sur cette machine, chaque programme serveur écoute sur un numéro de **port**.

Dans le navigateur, on écrit l'adresse puis le port après deux-points :

`http://127.0.0.1:5000`. L'adresse `127.0.0.1` désigne votre propre poste. Le port 5000 est celui qu'on utilise pendant le développement.

Quelques outils Python du labo

Le labo vous fait lire et compléter du code. Voici les mots qui reviennent.

Écrit dans le code

Ce que ça veut dire

`if ...: puis elif ...:`

« si » puis « sinon si » : on teste un autre cas.

`x is None`

vrai si `x` ne contient aucune valeur.

`"a" not in liste`

vrai si `"a"` n'est pas dans la liste.

`len(x)`

le nombre de caractères d'un texte, ou d'éléments d'une liste.

`x.strip()`

le texte `x` sans les espaces au début et à la fin.

`"5".isdigit()`

vrai si le texte ne contient que des chiffres.

`liste.append(valeur)`

ajoute `valeur` à la fin de la liste.

`try: ... except ValueError: ...`

« essaie ceci ; si ça rate avec cette erreur, fais cela ».

Vérifiez-vous

☐

1. Écrivez une fonction `bonjour(nom)` qui renvoie le texte « Bonjour » suivi du nom.

☐

2. Avec `c = {"id": 6, "statut": "ouvert"}`, que renvoie `c["statut"]` ?

☐

3. Une trace se termine par `NameError: name 'chantier' is not defined`. Quel est le problème ?

Réponses

1. `def bonjour(nom):` puis, en retrait, `return "Bonjour " + nom`.
2. Le texte `"ouvert"`.
3. Le code utilise une variable `chantier` qui n'a jamais reçu de valeur. Il faut chercher la ligne indiquée juste au-dessus.

Ce que fait le serveur, ce que fait le navigateur

COURS-U6-S2-05

Karim Benali dirige l'atelier de TechnoVert. Chaque soir, ses techniciens rentrent de chantier. Ils ont cinq minutes pour noter leur intervention, pas plus.

Aujourd'hui, ils la notent dans un tableur partagé. Un jour, un technicien a saisi sur un chantier déjà terminé. Personne ne l'a vu pendant trois semaines.

Karim veut une page web : on se connecte, on remplit, on envoie. Et si la saisie est fausse, la page doit refuser.

À la fin de la séance, vous saurez comment un serveur Python construit cette page, reçoit la saisie et la vérifie.

LE PLAN DE LA SÉANCE

- 1 Qui fait quoi : le serveur ou le navigateur.
- 2 Route, vue, gabarit : le chemin d'une page.
- 3 Recevoir une saisie et la vérifier.
- 4 Se souvenir de l'utilisateur : la session.
- 5 Quand le serveur plante.

LA QUESTION DE DÉPART

Vous remplissez un formulaire sur un site, puis vous cliquez sur « Envoyer ». Qui doit vérifier que ce que vous avez tapé est juste : votre navigateur ou le serveur ?

CAPSULE 1 SUR 5

Qui fait quoi : le serveur ou le navigateur

On part de ce que vous faites tous les jours : taper une adresse et voir une page. Mais qui fabrique cette page ?

Le navigateur envoie une **REQUÊTE** au serveur. C'est un court message qui dit : « donne-moi la page `/chantiers` ». Le serveur répond avec une page écrite en **HTML**, le langage qui décrit le contenu d'une page : titres, tableaux, formulaires.

Pensez à un fast-food. Vous passez commande au comptoir, la cuisine prépare, on vous tend le plateau. Vous ne voyez jamais la cuisine.

Le navigateur, c'est vous. Le serveur, c'est la cuisine.

- 1 Le navigateur **envoie une requête** : une adresse, parfois des données.
- 2 Le serveur **exécute du code Python** : il lit les données, fait les calculs.
- 3 Le serveur **renvoie du HTML** : une page toute prête.
- 4 Le navigateur **affiche** la page. Il applique le **CSS**, le fichier qui règle les couleurs, les tailles et les marges.

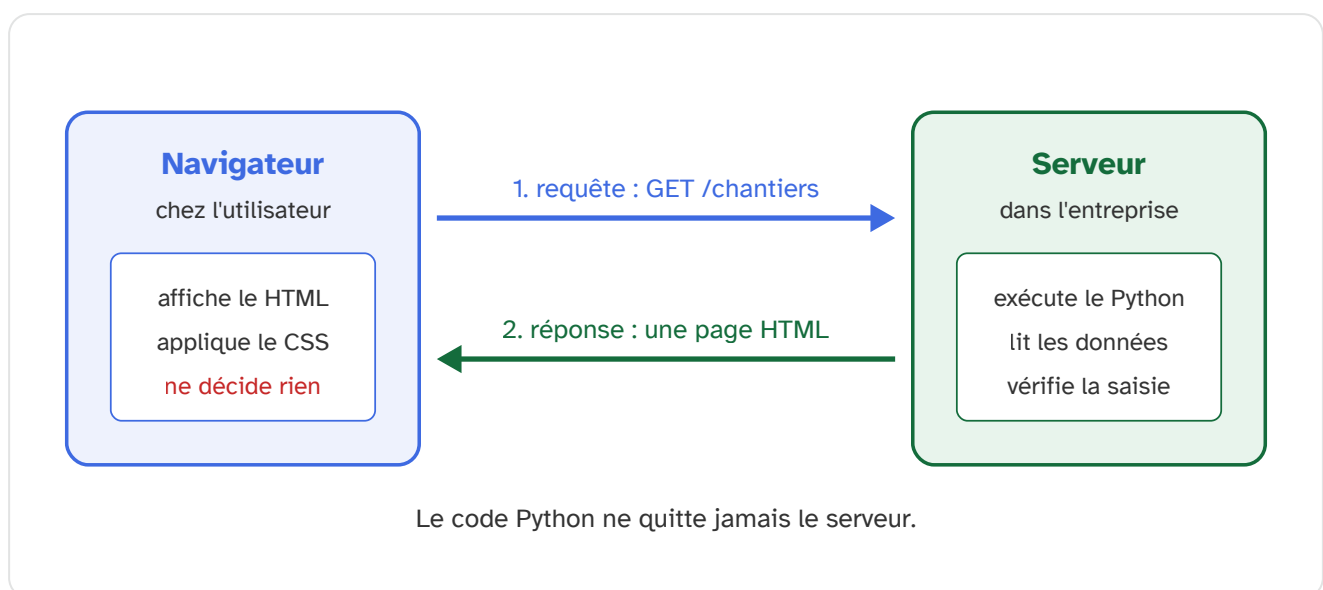


Schéma • Le navigateur demande, le serveur construit la page en Python et renvoie du HTML

Le HTML contient aussi le **formulaire** : les champs à remplir et le bouton « Envoyer ». Le navigateur l'affiche. Mais c'est le serveur qui reçoit et traite ce qu'on y tape.

ON LE FAIT ENSEMBLE — LA PAGE DE SAISIE DE KARIM

1. Qui envoie la requête quand un technicien ouvre la page de saisie ?
2. Où se trouve la liste des chantiers affichée dans le menu déroulant ?
3. Qui choisit la couleur verte du bandeau ?
4. Le technicien clique sur « Enregistrer ». Qui range l'intervention ?

Exemple	Pourquoi
✓ Le serveur lit le fichier des chantiers et met la liste dans la page.	Les données restent sur le serveur. Seul le résultat part vers le navigateur.
✗ « Le navigateur va chercher les chantiers dans le fichier. »	Le navigateur n'a pas accès aux fichiers du serveur. Il reçoit seulement du HTML.
✗ « Le CSS vérifie que la durée est un nombre. »	Le CSS règle l'apparence. Il ne vérifie rien et ne calcule rien.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Vous faites clic droit, puis « Afficher le code source » sur une page. Voyez-vous le code Python du serveur ?
2. Karim veut des lettres plus grandes sur la page. Faut-il changer le Python, le HTML ou le CSS ?

CAPSULE 2 SUR 5

Route, vue, gabarit : le chemin d'une page

Le serveur construit donc la page. Reste à savoir comment le code Python sait quelle page construire.

On utilise un **cadriciel web**, une boîte à outils qui gère les requêtes pour nous. Ici, c'est **Flask**, en Python. Avec Flask, une page passe par trois pièces : la **ROUTE**, la **VUE** et le **GABARIT**.

C'est comme au lycée. L'emploi du temps dit « mardi 10 h, salle B12 » : c'est la route.

Le professeur fait le cours : c'est la vue. Le manuel donne la mise en page : c'est le gabarit.

- 1 La **route** relie une adresse à une fonction : `/chantiers` mène à `liste_chantiers()`.
- 2 La **vue** est cette fonction Python. Elle va chercher les données.
- 3 La vue **passse les données** au gabarit, sous un nom : `chantiers=...`.

- 4 Le **gabarit** est un fichier HTML avec des trous. Il remplit les trous avec les données. Flask utilise pour ça **Jinja2**.

```
@app.route("/chantiers")           # la route
def liste_chantiers():             # la vue
    chantiers = donnees.lire_chantiers() # les données
    return render_template("chantiers.html", chantiers=chantiers)
```

Dans le gabarit, `{{ c.nom }}` affiche une valeur. `{% for c in chantiers %}` répète une ligne pour chaque chantier.

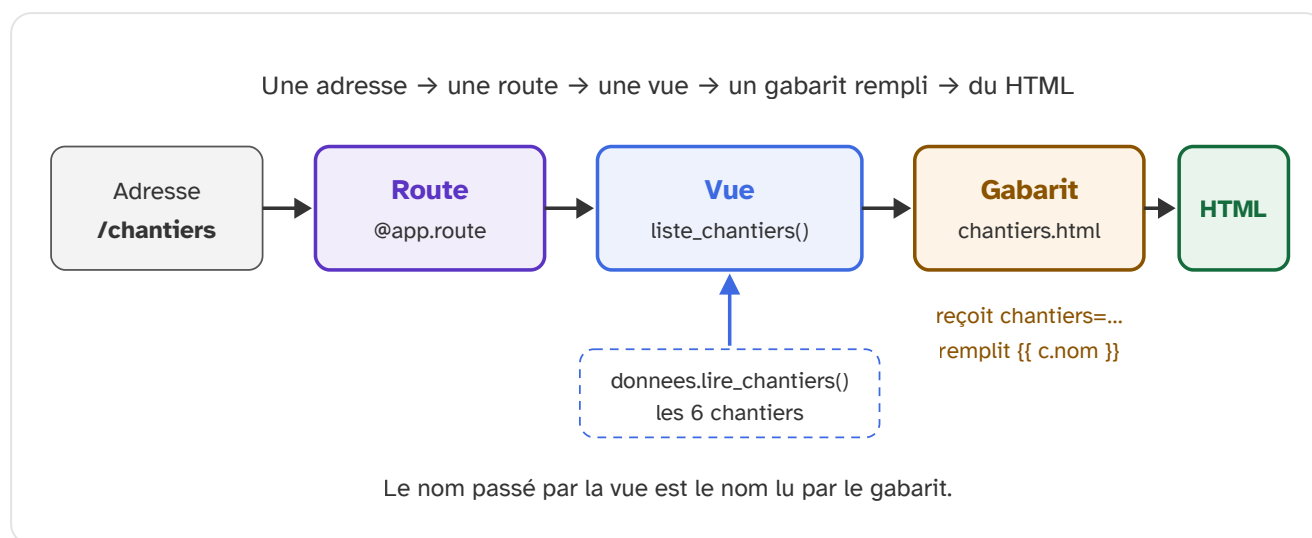


Schéma • De l'adresse /chantiers à la page affichée : route, vue, données, gabarit

ON LE FAIT ENSEMBLE — LA PAGE /SAISIR

1. Quelle ligne relie l'adresse `/saisir` à une fonction ?
2. Le gabarit `saisir.html` contient `{% for c in chantiers %}`. Quel nom la vue doit-elle utiliser pour passer la liste ?
3. La vue écrit `return render_template("saisir.html", liste=chantiers)`. Que voit-on ?

Exemple

✓ `render_template("chantiers.html",
chantiers=chantiers)`

Pourquoi

Le nom `chantiers` est celui que le gabarit attend.

Exemple	Pourquoi
❌ La vue fabrique le HTML avec des <code>+</code> et des <code>print</code> .	C'est illisible et fragile. Le HTML va dans le gabarit, le Python dans la vue.
❌ Deux fonctions sous la même route.	Flask n'appelle que la première. Une adresse, une vue.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Le navigateur demande `/connexion`. Quelle pièce Flask regarde-t-il en premier ?
2. Un gabarit affiche `{{ message }}`. Que doit écrire la vue pour afficher « Enregistré » ?

CAPSULE 3 SUR 5

Recevoir une saisie et la vérifier

La vue sait maintenant envoyer des données à la page. Il faut faire le chemin inverse : recevoir ce que le technicien a tapé.

Les données voyagent de deux façons. Avec **GET**, elles sont **dans l'adresse**, après un point d'interrogation. Avec **POST**, elles sont **dans le corps de la requête**, la partie invisible du message.

C'est comme une carte postale et une lettre. Sur la carte postale, tout le monde lit le texte. Dans l'enveloppe, le texte ne se voit pas de l'extérieur.

Attention : l'enveloppe cache, mais elle ne protège pas. Seul le chiffrement protège.

- 1 **Choisir** la méthode : GET pour chercher ou filtrer, POST pour envoyer une saisie.
- 2 **Lire** les données dans la vue : `request.args` pour GET, `request.form` pour POST.
- 3 **Vérifier chaque champ** : présent, du bon type, dans les bonnes limites.
- 4 **Répondre** : s'il y a une erreur, réafficher le formulaire avec un message. Sinon, enregistrer.

```
GET /chantiers?service=atelier → request.args["service"] vaut "atelier"
POST /saisir (corps : duree=2.5 ...) → request.form["duree"] vaut "2.5"
```

Tout ce qui arrive est du **texte**. "2.5" n'est pas encore un nombre : la vue doit le convertir, et refuser s'il n'y arrive pas.

GET : les données sont dans l'adresse

GET /chantiers ?service=atelier → la vue lit request.args

Visible dans la barre d'adresse, l'historique et le journal du serveur.

POST : les données sont dans le corps

POST /saisir

chantier=3 · duree=2.5 · description=... → la vue lit request.form

Dans les deux cas, tout arrive en texte : la vue vérifie chaque champ.

Schéma • Deux chemins pour les données : dans l'adresse avec GET, dans le corps avec POST

ON LE FAIT ENSEMBLE — UNE SAISIE DE DURÉE

1. Le technicien envoie le formulaire de saisie. GET ou POST ?
2. Où la vue lit-elle la durée ?
3. Le champ contient deux heures . Que fait la vue ?
4. Le champ contient 40 . Est-ce accepté ?

Exemple

Pourquoi

- | | |
|--|--|
| ✓ /chantiers?service=atelier pour filtrer la liste | Un filtre ne modifie rien. L'adresse peut être gardée en favori. |
| ✗ Un mot de passe envoyé en GET | Il apparaît dans l'adresse, dans l'historique et dans les journaux du serveur. |

Exemple

Pourquoi

✗ « Le champ a l'attribut `required` : pas besoin de vérifier sur le serveur. »

Cet attribut est dans le navigateur. On peut l'enlever en deux clics. Le serveur vérifie toujours.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. L'adresse `/chantiers?service=commercial` est demandée. Où la vue lit-elle `commercial` ?
2. Un chantier a un statut : `ouvert` ou `cloture`. On ne saisit plus rien sur un chantier clôturé. Le champ « chantier » contient le numéro d'un chantier clôturé. La saisie est-elle acceptée ?

CAPSULE 4 SUR 5

Se souvenir de l'utilisateur : la session

La saisie est vérifiée. Mais Karim veut une chose de plus : seuls les salariés connectés peuvent saisir. Et le serveur oublie tout entre deux requêtes.

Pour s'en souvenir, il y a la **SESSION**. C'est une petite mémoire liée à un visiteur.

Flask la range dans un **cookie**, un petit fichier que le navigateur renvoie à chaque requête. Le cookie est signé par une clé secrète du serveur : si quelqu'un le modifie, le serveur le refuse.

C'est le bracelet d'un festival. On le reçoit à l'entrée en montrant son billet. Ensuite, on le montre à chaque scène, sans ressortir le billet.

- 1 À la **connexion**, la vue vérifie l'identifiant et le mot de passe.
- 2 Si c'est bon, elle **écrit dans la session** : `session["login"] = login`.
- 3 Sur une page protégée, la vue **regarde la session** en premier. Pas de login ? Elle renvoie vers la page de connexion.
- 4 À la **déconnexion**, la vue **vide la session** : `session.clear()`.

Renvoyer vers une autre page s'appelle une **REDIRECTION**. Le serveur ne donne pas de page : il répond « va voir à cette adresse ». Chaque réponse porte un **code de réponse** :

Code	Sens	Exemple dans le mini-site
200	Tout va bien, voici la page.	La liste des chantiers s'affiche.
302	Redirection : va voir ailleurs.	Pas connecté : renvoyé vers <code>/connexion</code> .
404	Cette adresse n'existe pas.	<code>/chantiers/99</code> quand il n'y a pas de chantier 99.
500	Le code du serveur a planté.	Une erreur Python dans une vue.



Schéma • La connexion écrit le login dans la session ; chaque page protégée le vérifie avant de répondre

ON LE FAIT ENSEMBLE — YANN ROPARS OUVRE /SAISIR SANS ÊTRE CONNECTÉ

1. Que regarde la vue en premier ?
2. Il n'y en a pas. Quel code le serveur renvoie-t-il ?
3. Yann se connecte avec le bon mot de passe. Qu'écrit la vue ?
4. Il rouvre `/saisir`. Que se passe-t-il ?

Exemple	Pourquoi
✓ Après une connexion réussie, une redirection vers la liste	L'utilisateur arrive sur une vraie page, et un rechargement ne renvoie pas le mot de passe.

Exemple	Pourquoi
❌ Protéger la page en cachant le lien dans le menu	L'adresse reste joignable. Il faut vérifier la session dans la vue.
❌ Mettre le mot de passe dans la session	Inutile et risqué. Le login suffit : le mot de passe a déjà été vérifié.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Où Flask range-t-il la session ?
2. Karim clique sur « Se déconnecter », puis rouvre l'adresse `/saisir` à la main. Peut-il saisir ?

CAPSULE 5 SUR 5

Quand le serveur plante

Une vue plante : une clé manque, un calcul échoue. Le serveur répond alors avec le code 500. Mais que voit l'utilisateur ?

Tout dépend du **mode** du serveur. En **mode développement**, Flask affiche la trace complète dans le navigateur : le code, les fichiers, les variables. C'est pratique pour le développeur.

En **mode production**, le navigateur reçoit seulement « Internal Server Error ». La trace part dans le **journal du serveur**, là où le technicien la lit.

Mode	Ce que voit le navigateur	Où lire la trace
Développement : <code>flask run --debug</code>	La trace complète, le code source	Dans la page et dans le terminal
Production : <code>flask run</code> sans <code>--debug</code>	« Internal Server Error », rien d'autre	Dans le terminal du serveur, ou son journal

La règle : **la trace ne doit jamais atteindre le navigateur d'un utilisateur**. Elle montre le code, les chemins des fichiers, parfois des mots de passe. Et le mode développement de Flask peut même laisser exécuter du code Python depuis la page.

INSTANT HISTOIRE — 2015 : PATREON ET LE MODE DÉVELOPPEMENT

En 2015, Patreon est un site qui aide des artistes à se faire financer par leur public.

Une version de développement du site est restée joignable depuis Internet, avec le mode de débogage allumé.

Des pirates s'en servent pour entrer. Les données de près de 2,3 millions de comptes sont volées puis publiées.

Qu'est-ce qui aurait évité ça ?

Ce qu'on retient

SI VOUS NE DEVEZ RETENIR QUE TROIS CHOSES

1. Le serveur construit la page en Python et envoie du HTML. Le navigateur affiche, il ne décide rien.
2. Chaque saisie se vérifie dans la vue, sur le serveur, champ par champ. Même si le navigateur a déjà vérifié.
3. Une page protégée vérifie la session en premier. Et en production, la trace d'erreur reste dans le journal.

Revenons à Karim. Son technicien ouvre `/saisir` : sans session, il est renvoyé vers la connexion.

Connecté, il choisit un chantier et envoie le formulaire en POST. La vue voit que le chantier est clôturé, et elle refuse avec un message. L'erreur du tableur ne peut plus passer.

Ce que fait le serveur, ce que fait le navigateur

COURS-U6-S2-05

L'essentiel

- **Serveur et navigateur** : le serveur exécute le Python et renvoie du HTML. Le navigateur affiche, avec le CSS.
- **Route, vue, gabarit** : la route relie une adresse à une vue. La vue lit les données et les passe au gabarit. Le gabarit remplit le HTML.
- **GET et POST** : GET met les données dans l'adresse, POST dans le corps de la requête.
- **Valider sur le serveur** : chaque champ, à chaque fois. Le navigateur peut être contourné.
- **Session** : une mémoire par visiteur, dans un cookie signé. Une page protégée la vérifie en premier.
- **Production** : pas de mode débogage. La trace va dans le journal, jamais dans le navigateur.

Les mots à connaître

Mot	En clair
Requête	Le message du navigateur au serveur : une adresse, parfois des données.
HTML	Le langage qui décrit le contenu d'une page.
CSS	Le fichier qui règle couleurs, tailles et marges.
Route	Le lien entre une adresse et une fonction Python.
Vue	La fonction Python qui prépare la réponse.
Gabarit	Un fichier HTML avec des trous <code>{{ ... }}</code> , rempli par Jinja2.
Session	Ce que le serveur retient d'un visiteur entre deux requêtes.

Redirection Réponse 302 : « va voir à cette adresse ».

La méthode : écrire une vue qui reçoit un formulaire

1. **Vérifiez** la session. Pas de login : redirection vers `/connexion`.
2. **Distinguez** GET et POST avec `request.method`.
3. **Lisez** chaque champ dans `request.form`.
4. **Vérifiez** chaque champ : présent, bon type, bonnes limites. Notez chaque erreur dans une liste.
5. **Répondez** : erreurs → réafficher avec les messages. Sinon → enregistrer, puis afficher la confirmation.

Un exemple concret

Un technicien envoie une durée de `abc` sur le chantier 3.

Étape	Ce qu'on fait	Résultat
1	<code>"login" in session</code>	Oui : on continue.
2	<code>request.method</code>	<code>"POST"</code> : c'est un envoi.
3	<code>request.form["duree"]</code>	Le texte <code>"abc"</code> .
4	<code>float("abc")</code>	Échec : on ajoute « La durée doit être un nombre ».
5	Réafficher <code>saisir.html</code> avec <code>erreurs</code>	Le message s'affiche. Rien n'est enregistré.

Conclusion : la saisie fausse est refusée par le serveur, avec un message clair.

Les pièges

- **✗ Faire confiance au navigateur** → toujours vérifier dans la vue.
- **✗ Un nom différent dans la vue et dans le gabarit** → `chantiers=` des deux côtés.

- **✗ Oublier que tout arrive en texte** → convertir avec `int()` ou `float()`, et prévoir l'échec.
- **✗ Laisser `--debug` en production** → le lancer seulement sur votre poste de travail.

Un mini-site à trois pages

LABO-U6-S2-05

Contexte — Vous travaillez pour la société qui reprend l'informatique de TechnoVert. Karim Benali, le responsable d'atelier, veut que ses techniciens saisissent leurs interventions sur une page web, et plus sur un tableau.

Un collègue a préparé le squelette du site : les adresses, les pages HTML et la feuille de style. Il reste à écrire le code Python qui fait tourner chaque page.

RÈGLE 1A DE LA SÉANCE

Scénario 2 : l'IA peut vous expliquer une notion ou un message d'erreur. Elle n'écrit pas votre code.

Si vous lui posez une question, dites-la au formateur au point de contrôle suivant.

À la fin de ce labo, vous saurez écrire les vues d'un site Flask. Elles affichent une liste, traitent un formulaire et protègent une page. Vous saurez aussi lire une erreur serveur.

Comment ça se valide — à chaque point de contrôle, vous montrez votre résultat au formateur. Il valide, puis vous passez à la suite.

Les mots de la séance

Mot	Ce que ça veut dire
Route	La ligne <code>@app.route(...)</code> qui relie une adresse à une fonction.
Vue	La fonction Python placée sous la route. C'est elle que vous écrivez.
Gabarit	Le fichier HTML avec des trous <code>{{ ... }}</code> , dans le dossier <code>templates</code> . Il est fourni.
GET, POST	Deux façons d'envoyer des données : dans l'adresse, ou dans le corps de la requête.
Session	Ce que le serveur retient d'un visiteur, comme son identifiant de connexion.
Trace	Le message que Python affiche quand il plante. On le lit par le bas.

Ce dont vous avez besoin

- Votre poste Windows, avec Python, Flask, VS Code et un navigateur.
- [Annexe A1](#) : le mini-site de départ et son mode d'emploi. Le dossier se télécharge depuis l'[annexe A1](#), dans le fichier `A1_mini_site.zip`.
- [Annexe A2](#) : les gabarits du mini-site.
- [Annexe A3](#) : les données de TechnoVert et les comptes de test.
- [Annexe A4](#) : un extrait de la documentation de Flask, avec les conditions dans les gabarits.
- [Annexe A2](#) : des gabarits complets, à prendre comme modèles.

En binôme — le pilote tape, le lecteur lit les consignes et les annexes à voix haute, et vérifie chaque résultat. On inverse à la partie C.

Le plan du labo

Partie	Étapes	Ce que vous obtenez	Temps
A — Afficher la liste	1 à 6	la page des chantiers remplie par le serveur	25 min
B — Recevoir un formulaire	7 à 12	une saisie enregistrée par le serveur	30 min
C — Vérifier et protéger	13 à 18	des saisies vérifiées, des pages réservées aux salariés connectés	40 min
D — Lire une erreur serveur	19 à 22	une erreur lue au bon endroit, et le mode production	20 min

PARTIE A — Afficher la liste

Étapes 1 à 6, environ 25 min. À la fin, la page `/chantiers` affiche les six chantiers de TechnoVert.

Besoin d'un rappel ? Revoir le cours : [« Route, vue, gabarit : le chemin d'une page »](#).

01 Installer le mini-site sur votre poste

- ☐ Téléchargez `A1_mini_site.zip` depuis l'[annexe A1](#).
- ☐ Faites un clic droit sur le fichier, puis « Extraire tout ». Avec « Parcourir », choisissez votre dossier « Documents ».

Vous devez voir : un dossier `mini-site` dans « Documents ». Il contient `app.py`, `chantiers.json`, `donnees.py`, et les dossiers `static` et `templates`.

02 Ouvrir le dossier dans VS Code

VS Code sert à la fois à modifier le code et à lancer le serveur.

- ☐ Ouvrez VS Code. Menu « Fichier », puis « Ouvrir le dossier ». Choisissez `Documents\mini-site`.
- ☐ Ouvrez un terminal : menu « Terminal », puis « Nouveau terminal ». Tapez :

```
ls
```

Vous devez voir : la liste des fichiers du dossier, dont `app.py` et `templates`.

Si vous voyez une liste qui ne contient pas `app.py` : vous avez ouvert le mauvais dossier. Ouvrez celui qui contient directement `app.py`.

03 Lancer le serveur en mode développement

☐ Dans le terminal, tapez :

```
python -m flask --app app run --debug
```

Les options de cette commande sont expliquées dans l'[annexe A1](#).

Vous devez voir : entre autres ces lignes.

```
* Running on http://127.0.0.1:5000
* Debugger is active!
```

Si vous voyez « python n'est pas reconnu » : tapez `py` à la place de `python`.

☐ Dans votre navigateur, ouvrez `http://127.0.0.1:5000/chantiers`.

Vous devez voir : le texte « Page des chantiers : à écrire ». La route existe, mais sa vue n'est pas écrite.

Ce terminal reste ouvert jusqu'à la partie D. En mode développement, le serveur recharge seul le code à chaque enregistrement.

04 Lire ce que le gabarit attend

Un gabarit a des trous. La vue doit lui donner les bonnes données, sous le bon nom.

☐ Ouvrez l'[annexe A2](#) et lisez le gabarit `chantiers.html`.

☐ Notez le nom de la liste que le gabarit parcourt dans sa boucle `for` :

☐ Notez les cinq clés que le gabarit lit pour chaque chantier :

Vous devez avoir : un nom de liste, et cinq clés. Vous les retrouverez dans le code de l'étape 5.

05 Écrire la vue de la liste

Voici la vue complète. Elle lit les chantiers, puis les passe au gabarit sous le nom attendu.

☐ Dans VS Code, ouvrez `app.py` depuis la liste des fichiers, à gauche.

☐ Remplacez le corps de `liste_chantiers()` par ces deux lignes, avec quatre espaces devant chacune :

```
chantiers = donnees.lire_chantiers()
return render_template("chantiers.html", chantiers=chantiers)
```

☐ Enregistrez avec Ctrl+S. Rechargez la page dans le navigateur.

Vous devez voir : le titre « Les chantiers », la phrase « 6 chantier(s). » et un tableau de six lignes. La première est « Arrosage du parc de la mairie ».

Si vous voyez « IndentationError » dans la page : une ligne n'a pas ses quatre espaces devant. Rouvrez le fichier et alignez-la.

06 Vérifier que la page vient du serveur

☐ Dans le navigateur, affichez le code source de la page : Ctrl+U.

☐ Cherchez le mot `lire_chantiers` dans le code source avec Ctrl+F.

Vous devez voir : aucun résultat. Le code source contient seulement du HTML, avec une balise `<tr>` par chantier.

Pourquoi ? Pourquoi le navigateur ne reçoit-il jamais le code Python de la vue ?

Votre réponse :

POINT DE CONTRÔLE A

☐ Montrez au formateur la liste des six chantiers, et votre réponse à l'étape 6.

☐ Le formateur a validé. Passez à la partie B.

PARTIE B — Recevoir un formulaire

Étapes 7 à 12, environ 30 min. À la fin, une intervention saisie dans la page est enregistrée par le serveur.

Besoin d'un rappel ? Revoir le cours : « [Recevoir une saisie et la vérifier](#) ».

07 Afficher le formulaire vide

La page `/saisir` reçoit deux sortes de requêtes. En GET, on affiche le formulaire vide. En POST, on reçoit la saisie.

☐ Ouvrez l'[annexe A2](#) et lisez la liste des données qu'attend le gabarit `saisir.html`.

☐ Dans `app.py`, remplacez le corps de `saisir()` par :

```
chantiers = donnees.lire_chantiers()
if request.method == "GET":
    return render_template("saisir.html", chantiers=chantiers,
services=donnees.SERVICES,
                                erreurs=[], saisie={}, message=None)
```

☐ Enregistrez, puis ouvrez `/saisir` dans le navigateur.

Vous devez voir : le formulaire, avec six chantiers dans le premier menu et trois services dans le second.

08 Trouver dans la doc

Il faut maintenant lire ce que le technicien a tapé. La méthode est dans la documentation.

☐ Ouvrez l'[annexe A4](#).

☐ Trouvez l'objet qui contient les champs envoyés dans le corps d'une requête POST. Notez son nom : _____

- ☐ Trouvez l'objet qui contient les données envoyées dans l'adresse. Notez son nom :

Vous devez avoir : deux noms qui commencent par `request.` . Le premier sert à l'étape 9.

09 Enregistrer la saisie

Pour l'instant, il n'y a pas de connexion. Le technicien s'appelle donc « test ». La partie C remplacera ce nom par le vrai.

- ☐ Ajoutez ces lignes à la fin de `saisir()` , sous le bloc `if` de l'étape 7 :

```
saisie = request.form
numero = donnees.ajouter_intervention(int(saisie["chantier"]), "test",
saisie["service"],
                                saisie["date"],
float(saisie["duree"]),
                                saisie["description"],
saisie["blocage"])
    return render_template("saisir.html", chantiers=chantiers,
services=donnees.SERVICES,
                                erreurs=[], saisie={}, message=f"Intervention n°
{numero} enregistrée.")
```

- ☐ Enregistrez le fichier.

Vous devez voir : la page `/saisir` s'affiche toujours. Aucune trace d'erreur dans le premier terminal.

10 Envoyer une saisie juste

- ☐ Dans le formulaire, choisissez le chantier 3, le service « atelier », la date `2026-09-29`, la durée `2.5` et la description « Contrôle des bornes ». Laissez le blocage vide. Cliquez sur « Enregistrer ».

Vous devez voir : le message vert « Intervention n° 13 enregistrée. ».

- ☐ Ouvrez un second terminal avec le bouton « + » du panneau Terminal. Le premier fait toujours tourner le serveur.
- ☐ Vérifiez que le fichier de données a changé :

```
Select-String "Contrôle des bornes" chantiers.json
```

Vous devez voir : une ligne qui contient `"description": "Contrôle des bornes"`, précédée du nom du fichier et du numéro de la ligne.

11 Voir la différence entre GET et POST

- ☐ Regardez la barre d'adresse du navigateur, juste après votre envoi. Notez l'adresse affichée : _____
- ☐ Ouvrez maintenant `/chantiers?service=atelier`.
- ☐ Dans le premier terminal, cherchez dans le journal du serveur la ligne `POST /saisir` et la ligne `GET /chantiers?service=atelier`. Ignorez les lignes `/static/style.css` et `/favicon.ico`. Elles ressemblent à :

```
127.0.0.1 - - [29/Sep/2026 14:02:11] "POST /saisir HTTP/1.1" 200 -  
127.0.0.1 - - [29/Sep/2026 14:03:40] "GET /chantiers?service=atelier HTTP/1.1" 200  
-
```

Vous devez voir : la saisie n'apparaît nulle part dans l'adresse ni dans le journal. Le mot `atelier`, lui, apparaît dans les deux.

Pourquoi ? Un collègue propose d'envoyer le mot de passe de connexion en GET. Pourquoi est-ce une mauvaise idée ?

Votre réponse :

12 Envoyer une saisie fausse

- ☐ Remplissez le formulaire avec le chantier 3, le service « atelier », la date `2026-09-29` et la durée `deux heures`. Cliquez sur « Enregistrer ».

Vous devez voir : une page d'erreur de Flask. Tout en bas, la ligne `ValueError: could not convert string to float: 'deux heures'`.

- ☐ Notez la ligne de `app.py` indiquée juste au-dessus de cette dernière ligne :

Pourquoi ? Le serveur a planté au lieu d'afficher un message. Qu'est-ce qui manque dans la vue ?

Votre réponse :

POINT DE CONTRÔLE B

- ☐ Montrez au formateur la ligne trouvée à l'étape 10, et vos réponses aux étapes 11 et 12.
- ☐ Le formateur a validé. Passez à la partie C.

PARTIE C — Vérifier et protéger

Étapes 13 à 18, environ 40 min. À la fin, les saisies fausses sont refusées avec un message, et seuls les salariés connectés peuvent saisir.

Besoin d'un rappel ? Revoir le cours : [« Se souvenir de l'utilisateur : la session »](#).

Changez de rôle : le lecteur devient pilote.

13 Vérifier la durée et le chantier

On vérifie chaque champ avant d'enregistrer. Chaque erreur trouvée va dans une liste. Voici les deux premières vérifications, écrites en entier.

- ☐ Dans `saisir()`, remplacez les lignes de l'étape 9, de `saisie = request.form` jusqu'à la fin, par :

```

saisie = request.form
erreurs = []

chantier = None
if saisie["chantier"].isdigit():
    chantier = donnees.lire_chantier(int(saisie["chantier"]))
if chantier is None:
    erreurs.append("Choisissez un chantier.")
elif chantier["statut"] == "cloture":
    erreurs.append("Ce chantier est clôturé : saisie refusée.")

duree = 0
try:
    duree = float(saisie["duree"])
    if duree <= 0 or duree > 12:
        erreurs.append("La durée doit être entre 0 et 12 heures.")
except ValueError:
    erreurs.append("La durée doit être un nombre, par exemple 2.5.")

# étape 14 : vos trois vérifications ici

if erreurs:
    return render_template("saisir.html", chantiers=chantiers,
services=donnees.SERVICES,
                                erreurs=erreurs, saisie=saisie, message=None)

numero = donnees.ajouter_intervention(chantier["id"], "test",
saisie["service"],
                                saisie["date"], duree,
                                saisie["description"],
saisie["blocage"])
    return render_template("saisir.html", chantiers=chantiers,
services=donnees.SERVICES,
                                erreurs=[], saisie={}, message=f"Intervention n°
{numero} enregistrée.")

```



Enregistrez, puis renvoyez la saisie fausse de l'étape 12.

Vous devez voir : le formulaire revient, avec le message rouge « La durée doit être un nombre, par exemple 2.5. ». Les champs déjà remplis sont gardés.

14 Écrire les trois autres vérifications

Chaque vérification tient en deux lignes : un `if`, puis `erreurs.append(...)`. Voici la première, écrite en entier. Elle refuse un service qui n'est pas dans la liste.

```
if saisie["service"] not in donnees.SERVICES:
    erreurs.append("Choisissez un service dans la liste.")
```

- ☐ Sur ce modèle, écrivez les deux autres. Aidez-vous du tableau « Quelques outils Python du labo » des RAPPELS : `len(x)` donne la longueur d'un texte, `x.strip()` enlève les espaces.

Champ	Le test à écrire	Message à afficher
date	le texte ne fait pas 10 caractères : <code>len(saisie["date"]) != 10</code>	La date s'écrit AAAA-MM-JJ.
description	le texte sans espaces est vide : <code>saisie["description"].strip() == ""</code>	Décrivez ce qui a été fait.

- ☐ Écrivez vos trois vérifications dans `app.py`, à la place du commentaire `# étape 14`, puis enregistrez.

Vous devez voir : la page `/saisir` s'affiche toujours, sans trace d'erreur.

15 Tester les vérifications

- ☐ Envoyez chaque saisie du tableau. Pour chacune, notez ce que la page affiche.

Saisie	Résultat attendu	Ce que vous obtenez
chantier 1, atelier, 2026-09-29 , 3 , « Contrôle »	refusée : chantier clôturé	
chantier 4, aucun service, 2026-09-29 , 3 , « Contrôle »	refusée : service	
chantier 4, atelier, 29/09 , 3 , « Contrôle »	refusée : date	
chantier 4, atelier, 2026-09-29 , 40 , description vide	refusée : deux messages	
chantier 4, atelier, 2026-09-29 , 3 , « Contrôle des mâts »	acceptée	

Vous devez voir : exactement le résultat attendu sur chaque ligne. La dernière affiche un numéro d'intervention.

Pourquoi ? Le menu déroulant ne propose que trois services. Pourquoi vérifier quand même le service sur le serveur ?

Votre réponse :

16 Écrire la connexion

La connexion vérifie le compte, puis écrit le login dans la session. La fonction `verifier_compte()` est fournie : [l'annexe A1](#) la décrit.

☐

Remplacez le corps de `connexion()` par :

```
if request.method == "POST":
    login = request.form["login"]
    if donnees.verifier_compte(login, request.form["mot_de_passe"]):
        session["login"] = login
        return redirect(url_for("liste_chantiers"))
    return render_template("connexion.html", erreur="Identifiant ou mot de
passe incorrect.")
return render_template("connexion.html", erreur=None)
```

- ☐ Remplacez le corps de `deconnexion()` par ces deux lignes :

```
session.clear()
return redirect(url_for("connexion"))
```

- ☐ Ouvrez `/connexion` et connectez-vous avec un compte de l'[annexe A3](#).

Vous devez voir : vous arrivez sur la liste des chantiers. Le bandeau affiche « Connecté : » suivi de votre identifiant.

17 Protéger les deux pages

Une page protégée regarde la session avant tout le reste.

- ☐ Ajoutez ces deux lignes au tout début de `liste_chantiers()` et au tout début de `saisir()`, avant toute autre ligne du corps :

```
if "login" not in session:
    return redirect(url_for("connexion"))
```

- ☐ Dans `saisir()`, dans l'appel à `ajouter_intervention`, remplacez `"test"` par `session["login"]`. Enregistrez.

Vous devez voir : tant que vous êtes connecté, les deux pages s'affichent comme avant.

18 Tester la session

- ☐ Faites chaque essai du tableau, dans l'ordre. Après chaque essai, cherchez dans le journal du premier terminal la ligne de la requête que vous venez de faire : `GET /saisir` ou `POST /connexion`. Relevez le code de trois chiffres à la fin de cette ligne. Les lignes qui suivent viennent du navigateur, qui charge la page suivante, `style.css` et `favicon.ico` : ignorez-les. Attention : une connexion ratée renvoie 200, car la page est réaffichée avec le message.

Essai	Résultat attendu	Code relevé
Cliquez sur « Se déconnecter », puis ouvrez <code>/saisir</code>	renvoyé vers la connexion	
Connectez-vous avec un mauvais mot de passe	message rouge, pas de connexion	
Connectez-vous avec le bon mot de passe	arrivée sur la liste	
Ouvrez <code>/saisir</code>	le formulaire s'affiche	

- ☐ Ouvrez les outils du navigateur avec F12. Dans l'onglet « Application » ou « Stockage », ouvrez « Cookies ».

Vous devez voir : un cookie nommé `session`, avec une valeur en trois morceaux séparés par des points.

Pourquoi ? Un utilisateur modifie la valeur de ce cookie pour y mettre le login d'un autre salarié. Pourquoi le serveur refuse-t-il ce cookie ?

Votre réponse :

POINT DE CONTRÔLE C

- ☐ Montrez au formateur votre grille de l'étape 15 et votre tableau de l'étape 18.
- ☐ Le formateur a validé. Passez à la partie D.

PARTIE D — Lire une erreur serveur

Étapes 19 à 22, environ 20 min. À la fin, vous savez où lire une erreur, et ce que voit l'utilisateur en production.

Besoin d'un rappel ? Revoir le cours : [« Quand le serveur plante »](#).

19 Lire l'erreur dans le navigateur

Un collègue a écrit une page `/bilan` qui additionne les heures. Elle plante.

- ☐ Ouvrez `/bilan` dans le navigateur.
- ☐ Lisez la trace par le bas. Notez le type d'erreur et la valeur entre guillemets :

- ☐ Notez le numéro de la ligne de `app.py` qui plante : _____

Vous devez voir : une page d'erreur de Flask, avec des morceaux du code de `app.py`.

20 Lire la même erreur dans le journal

- ☐ Regardez le premier terminal, celui du serveur.
-

Vous devez voir : la même trace, et une ligne qui se termine par
`"GET /bilan HTTP/1.1" 500 - .`

- ☐ Notez le code de réponse de cette ligne : _____
-

21 Passer en mode production

- ☐ Dans le premier terminal, arrêtez le serveur avec Ctrl+C.
-

- ☐ Relancez-le sans le mode développement :
-

```
python -m flask --app app run
```

- ☐ Rechargez `/bilan` dans le navigateur.
-

Vous devez voir : seulement « Internal Server Error », et une phrase en anglais. Aucune ligne de code. La trace complète est toujours dans le terminal.

Pourquoi ? Pourquoi l'utilisateur ne doit-il pas voir la trace de l'étape 19 ?

Votre réponse :

22 Trouver dans la doc et corriger

La ligne fautive lit une clé qui n'existe pas. Le nom juste est dans la description des données.

☐ Ouvrez l'[annexe A3](#). Trouvez le nom exact de la clé qui contient la durée d'une intervention. Notez-le : _____

☐ Corrigez la ligne de `bilan()` avec ce nom. Enregistrez.

☐ En mode production, le serveur ne recharge pas le code seul. Arrêtez-le avec Ctrl+C, relancez-le, puis rechargez `/bilan`.

Vous devez voir : la phrase « Total des heures saisies : » suivie d'un nombre d'heures.

POINT DE CONTRÔLE D

☐ Montrez au formateur la page `/bilan` corrigée, et vos réponses aux étapes 19 à 21.

☐ Le formateur a validé. Le labo est fini.

POUR ALLER PLUS LOIN

10 à 15 min, si vous avez fini avant la fin de la séance.

Un collègue a demandé à une IA d'écrire une page de connexion. Voici ce qu'elle a produit. Le code marche, mais il contient trois erreurs dangereuses.

```
@app.route("/connexion")
def connexion():
    login = request.args.get("login")
    mot_de_passe = request.args.get("mot_de_passe")
    if login and donnees.verifier_compte(login, mot_de_passe):
        session["login"] = login
        session["mot_de_passe"] = mot_de_passe
        return redirect(url_for("liste_chantiers"))
    return render_template("connexion.html", erreur=None)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

☐ Trouvez les trois erreurs avant la mise en production. Pour chacune, notez la ligne et le danger.

Ligne	L'erreur	Le danger

CE QU'IL FAUT RETENIR

1. La vue passe les données au gabarit sous le nom exact que le gabarit attend.
2. Chaque champ reçu est du texte : on le vérifie, on le convertit, et on prévoit l'échec.
3. Une page protégée vérifie la session en premier. En production, pas de `--debug` : la trace reste dans le journal.

L'interface de consultation des chantiers

TP-U6-S2-03

Contexte — Karim Benali vous appelle : « Mes techniciens veulent voir leurs chantiers depuis un navigateur. Pas le tableur, une vraie page. »

« Et Yann, notre commercial, veut voir ce que chaque service a fait chez ses clients. »

La base de données n'est pas encore branchée. Les données sont dans un fichier. Seuls les salariés connectés doivent voir les chantiers.

RÈGLE IA DE LA SÉANCE

Scénario 2 : l'IA peut vous expliquer une notion ou un message d'erreur. Elle n'écrit ni vos vues ni vos gabarits.

Notez dans votre mini-rapport chaque question posée à une IA.

CE QUE VOUS RENDEZ

Ce que vous rendez	Format	Nom du fichier
Le dossier <code>consultation</code> , complet	ZIP	NOM_Prenom_TP-U6-S2-03_appli.zip
Le mini-rapport	PDF	NOM_Prenom_TP-U6-S2-03_rapport.pdf

Le mini-rapport : deux pages au maximum. Il prouve que c'est **votre** travail : chaque rubrique s'appuie sur ce qui s'est passé sur **votre** poste. Un texte général, sans traces de votre poste, ne compte pas.

1. **Mes captures** : les quatre captures des parties A à D. Prenez-les en plein écran : l'heure de la barre des tâches doit se voir. Une ligne de légende sous chacune.
2. **Mon journal** : connecté, faites ce parcours dans l'ordre : `/chantiers`, filtre « atelier », `/chantiers/2`, `/chantiers/99`, « Se déconnecter », `/chantiers`. Recopiez les lignes de votre terminal, sauf `style.css` et `favicon.ico`. Pour chacune, notez le code de réponse et ce qu'il veut dire, en une phrase.

- 3. **Mes trois erreurs** : trois erreurs que vous avez vraiment rencontrées pendant le TP, dans le tableau ci-dessous.
- 4. **Une ligne de mon code** : recopiez une ligne de votre vue `liste()` . Expliquez-la avec vos mots, en trois lignes au maximum.
- 5. **Ma grille de lisibilité** : la grille de la partie D, remplie.
- 6. **Ce qui manque** : ce qui ne marche pas, ce que vous avez essayé, les coups de pouce utilisés, et les questions posées à une IA.

Voici une ligne d'exemple pour la rubrique 3 :

Le message d'erreur, recopié	Ce qui le causait	Ce que j'ai changé
UndefinedError: 'chantiers' is undefined	Ma vue passait <code>liste=</code> au gabarit, qui attend <code>chantiers</code> .	J'ai écrit <code>chantiers=chantiers</code> dans <code>render_template</code> .

Vos trois erreurs :

Le message d'erreur, recopié	Ce qui le causait	Ce que j'ai changé

Comment ce sera évalué

Critère	Ce qui est attendu
Les vues et les routes	Les trois écrans marchent, le filtre aussi, et les pages sont protégées par la session.
La présentation de l'interface	Les pages respectent la charte de lisibilité de l' annexe A7 .
Présentation	Document daté, nommé selon la convention, titres clairs, tableaux lisibles

Sans traces de votre poste, c'est-à-dire sans vos captures, votre journal ni vos erreurs, les deux premiers critères ne dépassent pas « Fragile ».

Et à l'oral — le formateur pointerait une ligne de votre code ou de votre mini-rapport. Il vous demanderait de l'expliquer, ou de refaire devant lui une erreur que vous racontez.

Ce dont vous disposez

- Votre poste Windows, avec Python, Flask, VS Code et un navigateur.
- [Annexe A5](#) : le point de départ du TP. Le dossier se télécharge depuis l'[annexe A5](#), dans le fichier [A5_consultation.zip](#).
- [Annexe A3](#) : les données de TechnoVert et les comptes de test.
- [Annexe A6](#) : les écrans attendus.
- [Annexe A7](#) : la charte de lisibilité minimale.
- [Annexe A4](#) : un extrait de la documentation de Flask, avec les conditions dans les gabarits.
- [Annexe A2](#) : des gabarits complets, à prendre comme modèles.

Le plan du TP

Partie	Ce que vous obtenez	Temps conseillé
A — La page de liste	la liste des chantiers	35 min
B — Le filtre par service	la liste filtrée par l'adresse	25 min
C — La page de détail	un chantier et ses interventions	35 min
D — La connexion et la lisibilité	des pages protégées et conformes à la charte	35 min
Mini-rapport		20 min

Gardez 20 minutes à la fin pour le mini-rapport.

Bloqué ? Demandez un coup de pouce. Ça ne fait pas perdre de points.

PARTIE A — La page de liste

Environ 35 min.

Vous construisez l'écran 1 de l'[annexe A6](#), sans filtre pour l'instant. Cette fois, vous écrivez la vue et le gabarit.

Besoin d'un rappel ? Revoir le cours : [« Route, vue, gabarit : le chemin d'une page »](#).

- ☐ 1. Lisez l'[annexe A5](#). Lancez le serveur en mode développement, dans le dossier `consultation`.
 - ☐ 2. Lisez l'écran 1 de l'[annexe A6](#). Notez les colonnes du tableau, dans l'ordre.
 - ☐ 3. Dans `app.py`, écrivez la route `/` : elle redirige vers la liste.
 - ☐ 4. Écrivez la route `/chantiers` et sa vue `liste()`. Elle lit les chantiers avec `donnees.lire_chantiers()` et les passe au gabarit `liste.html`.
 - ☐ 5. Créez le gabarit `templates/liste.html`. Il commence par `{% extends "base.html" %}`, comme `connexion.html`, et remplit le bloc `contenu`.
 - ☐ 6. Remplissez le contrat entre votre vue et votre gabarit :
-

Nom passé au gabarit	Ce qu'il contient	Où le gabarit l'utilise

Nom passé au gabarit	Ce qu'il contient	Où le gabarit l'utilise

Ce qui doit marcher à la fin de la partie A

- ☐ L'adresse `/` renvoie vers `/chantiers`.
- ☐ La page affiche le titre, le nombre de chantiers et un tableau de six lignes.
- ☐ Les colonnes sont celles de l'écran 1, dans le même ordre.
- ☐ Capture de la liste ajoutée au mini-rapport.

POINT DE CONTRÔLE A

- ☐ Le formateur a vu la liste des six chantiers. Passez à la partie B.

PARTIE B — Le filtre par service

Environ 25 min.

Yann veut voir les chantiers où un service est intervenu. Le choix du service passe dans l'adresse : `/chantiers?service=atelier`.

Besoin d'un rappel ? Revoir le cours : [« Recevoir une saisie et la vérifier »](#).

- ☐ 1. Relisez dans l'[annexe A5](#) ce que fait `lire_chantiers()` quand on lui donne un service.

☐ 2. Dans la vue `liste()`, lisez le service demandé dans l'adresse. Cherchez l'objet dans l'[annexe A4](#).

☐ 3. Écrivez la vérification : un service absent de `donnees.SERVICES` compte comme « tous les services ».

☐ 4. Ajoutez au gabarit le formulaire de filtre de l'écran 1. Sa méthode est GET.

☐ 5. Remplissez la grille de test :

Adresse demandée	Nombre de chantiers attendu	Nombre obtenu
<code>/chantiers</code>	6	
<code>/chantiers?service=atelier</code>	5	
<code>/chantiers?service=bureau etudes</code>	2	
<code>/chantiers?service=commercial</code>	3	
<code>/chantiers?service=nimporte</code>	6	

Ce qui doit marcher à la fin de la partie B

☐ Le menu propose « Tous les services » et les trois services.

☐ Après un filtre, le menu garde le service choisi.

☐ Chaque ligne de la grille donne le nombre attendu, sans trace d'erreur.

☐ Capture de la liste filtrée sur l'atelier ajoutée au mini-rapport.

POINT DE CONTRÔLE B

☐ Le formateur a vu la grille de test remplie. Passez à la partie C.

PARTIE C — La page de détail

Environ 35 min.

Un clic sur le nom d'un chantier ouvre sa page. Elle montre le chantier et toutes ses interventions.

-
- ☐ 1. Lisez l'écran 2 de l'[annexe A6](#).
 - ☐ 2. Écrivez la route `/chantiers/<int:id_chantier>` et sa vue `detail(id_chantier)`. Cherchez dans l'[annexe A4](#) comment Flask passe le numéro à la vue.
 - ☐ 3. Si le chantier n'existe pas, la vue répond avec le code 404. Cherchez la fonction dans l'[annexe A4](#).
 - ☐ 4. Créez le gabarit `templates/detail.html`, sur le modèle de `liste.html`.
 - ☐ 5. Dans `liste.html`, faites du nom de chaque chantier un lien vers sa page de détail, avec `url_for`.
 - ☐ 6. Remplissez la grille de test :
-

Adresse demandée	Résultat attendu	Résultat obtenu
<code>/chantiers/2</code>	3 interventions, dont une avec le blocage « acces refuse »	
<code>/chantiers/6</code>	1 intervention, blocage « aucun »	
<code>/chantiers/99</code>	page « Not Found », code 404 dans le journal	
<code>/chantiers/abc</code>	page « Not Found », code 404 dans le journal	

Ce qui doit marcher à la fin de la partie C

-
- ☐ Chaque nom de chantier de la liste mène à sa page.
 - ☐ La page de détail affiche les informations et le tableau de l'écran 2.
-

☐ Un numéro inconnu donne une page 404, jamais une trace d'erreur.

☐ Capture du détail du chantier 2 ajoutée au mini-rapport.

POINT DE CONTRÔLE C

☐ Le formateur a vu la page du chantier 2 et la page 404. Passez à la partie D.

PARTIE D — La connexion et la lisibilité

Environ 35 min.

Les chantiers ne sont visibles que par les salariés connectés. Puis vous relisez vos pages avec la charte.

Besoin d'un rappel ? Revoir le cours : [« Se souvenir de l'utilisateur : la session »](#).

☐ 1. Complétez les vues `connexion()` et `deconnexion()`, en bas de `app.py`. Le gabarit `connexion.html` est fourni : l'[annexe A5](#) dit ce qu'il attend.

☐ 2. Protégez la liste et le détail : sans session, redirection vers `/connexion`.

☐ 3. Testez avec un compte de l'[annexe A3](#), puis déconnectez-vous et rouvrez `/chantiers/2`.

☐ 4. Relisez vos deux pages avec la charte de l'[annexe A7](#). Remplissez la grille, une ligne par règle :

Règle de la charte

Liste : respectée ?

Détail : respectée ?

Ce que vous avez corrigé

1

2

3

4

5

6

7

8

☐ 5. Corrigez vos gabarits pour chaque règle non respectée.

Ce qui doit marcher à la fin de la partie D

☐ Sans connexion, `/chantiers` et `/chantiers/2` renvoient vers la connexion.

☐ Un mauvais mot de passe affiche un message, sans connecter.

☐ Après la déconnexion, les deux pages sont de nouveau fermées.

☐ La grille de lisibilité est remplie, et chaque règle est respectée sur les deux pages.

☐ Capture de la page de connexion ajoutée au mini-rapport.

POINT DE CONTRÔLE D

☐ Le formateur a vu une redirection vers la connexion, puis la grille remplie. Rédigez le mini-rapport.

Tous les documents du module

A1 — Le mini-site de départ et son mode d'emploi

↓ Télécharger A1_mini_site.zip

Source : application écrite pour le module. Données : voir l'[annexe A3](#). Fichier

[A1_mini_site.zip](#) à télécharger ci-dessus. Utilisée dans : LABO étapes 2, 3, 5 à 17 et 19.

Ce que contient le dossier

```
mini-site/
  app.py          les routes et les vues : c'est le seul fichier que vous
modifiez
  donnees.py      les fonctions qui lisent et écrivent les données : fourni
  chantiers.json  les données de TechnoVert (annexe A3)
  templates/      les gabarits HTML (annexe A2) : fournis
  static/style.css la feuille de style : fournie
```

Le fichier app.py de départ

Les routes existent. Les vues sont à écrire aux endroits marqués « À ÉCRIRE ».

```

"""Mini-site de suivi des chantiers de TechnoVert.

Les routes et les gabarits sont fournis.
Vous écrivez le corps des fonctions de vue, aux endroits marqués « À ÉCRIRE ».
"""

from flask import Flask, abort, redirect, render_template, request, session,
url_for

import donnees

app = Flask(__name__)
app.secret_key = "cle-de-labo-a-changer"    # clé qui signe le cookie de session

@app.route("/")
def accueil():
    return redirect(url_for("liste_chantiers"))

# Page 1 : la liste des chantiers
@app.route("/chantiers")
def liste_chantiers():
    # À ÉCRIRE (partie A)
    return "Page des chantiers : à écrire"

# Page 2 : la saisie d'une intervention
@app.route("/saisir", methods=["GET", "POST"])
def saisir():
    # À ÉCRIRE (parties B et C)
    return "Page de saisie : à écrire"

# Page 3 : la connexion
@app.route("/connexion", methods=["GET", "POST"])
def connexion():
    # À ÉCRIRE (partie C)
    return render_template("connexion.html", erreur=None)

@app.route("/deconnexion")
def deconnexion():
    # À ÉCRIRE (partie C)
    return redirect(url_for("connexion"))

# Écrite par un collègue. Ne la corrigez pas avant l'étape 22.
@app.route("/bilan")
def bilan():
    interventions = donnees.lire_interventions()

```

```
total = sum(i["duree"] for i in interventions)
return f"Total des heures saisies : {total} h"
```

Les fonctions de donnees.py

Fonction	Ce qu'elle renvoie
<code>donnees.lire_chantiers()</code>	La liste des chantiers, triés par date de début. Chaque chantier est un dictionnaire. Ses clés : <code>id</code> , <code>id_client</code> , <code>client</code> , <code>nom</code> , <code>adresse</code> , <code>date_debut</code> , <code>date_fin_prevue</code> , <code>statut</code> .
<code>donnees.lire_chantiers(service)</code>	Seulement les chantiers où ce service est intervenu.
<code>donnees.lire_chantier(numero)</code>	Un chantier, ou <code>None</code> s'il n'existe pas. <code>numero</code> est un entier.
<code>donnees.lire_interventions(id_chantier=..., service=...)</code>	La liste des interventions. Elle est filtrée si on donne un chantier ou un service.
<code>donnees.ajouter_intervention(id_chantier, technicien, service, date, duree, description, blocage)</code>	Enregistre une intervention dans <code>chantiers.json</code> et renvoie son numéro.
<code>donnees.verifier_compte(login, mot_de_passe)</code>	<code>True</code> si le compte existe et si le mot de passe est juste, sinon <code>False</code> .
<code>donnees.SERVICES</code>	La liste des trois services : <code>["atelier", "bureau etudes", "commercial"]</code> .

Installer le dossier sur votre poste

1. Téléchargez [A1_mini_site.zip](#) depuis cette annexe.
2. Faites un clic droit sur le fichier, puis « Extraire tout ». Avec « Parcourir », choisissez le dossier « Documents ».
3. Vérifiez : le dossier `Documents\mini-site` contient directement `app.py`.

Pour repartir de zéro, supprimez le dossier `mini-site` et extrayez de nouveau le zip.

Lancer le serveur

Ouvrez le dossier `mini-site` dans VS Code, puis un terminal : menu « Terminal », « Nouveau terminal ». Tapez :

```
python -m flask --app app run --debug
```

- `python -m flask` : lance Flask avec le Python de votre poste. Si `python` n'est pas reconnu, tapez `py`.
- `--app app` : l'application est dans `app.py`.
- `--debug` : le mode développement. Sans cette option, le serveur est en mode production.

Le serveur écoute sur le port 5000 de votre poste. Ouvrez `http://127.0.0.1:5000/chantiers` dans le navigateur. On arrête le serveur avec Ctrl+C.

Si Flask manque sur le poste, installez-le une fois : `python -m pip install --user flask`.

A2 — Les gabarits du mini-site

Source : gabarits écrits pour le module, fournis dans [A1_mini_site.zip](#). Utilisée dans : LABO étapes 4 et 7.

Ce que chaque gabarit attend de sa vue

Gabarit	Donnée à passer	Ce qu'elle contient
<code>chantiers.html</code>	<code>chantiers</code>	la liste des chantiers
<code>saisir.html</code>	<code>chantiers</code>	la liste des chantiers, pour le menu
<code>saisir.html</code>	<code>services</code>	la liste des trois services, pour le menu
<code>saisir.html</code>	<code>erreurs</code>	la liste des messages d'erreur, vide s'il n'y en a pas
<code>saisir.html</code>	<code>saisie</code>	les champs déjà tapés, ou <code>{}</code> pour un formulaire vide
<code>saisir.html</code>	<code>message</code>	le message de réussite, ou <code>None</code>
<code>connexion.html</code>	<code>erreur</code>	le message d'erreur de connexion, ou <code>None</code>
<code>base.html</code>	rien	il lit la session pour afficher « Connecté : »

base.html

Tous les autres gabarits le complètent : ils remplissent ses blocs `titre` et `contenu`.

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>{% block titre %}Chantiers{% endblock %} · TechnoVert</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <header>
    <p class="marque">TechnoVert · Suivi des chantiers</p>
    <nav>
      <a href="{{ url_for('liste_chantiers') }}">Chantiers</a>
      <a href="{{ url_for('saisir') }}">Saisir une intervention</a>
      {% if session.get('login') %}
        <span class="qui">Connecté : {{ session['login'] }}</span>
        <a href="{{ url_for('deconnexion') }}">Se déconnecter</a>
      {% else %}
        <a href="{{ url_for('connexion') }}">Se connecter</a>
      {% endif %}
    </nav>
  </header>
  <main>
    {% block contenu %}{% endblock %}
  </main>
</body>
</html>
```

chantiers.html

```
{% extends "base.html" %}
{% block titre %}Chantiers{% endblock %}
{% block contenu %}
<h1>Les chantiers</h1>
<p>{{ chantiers|length }} chantier(s).</p>
<table>
  <thead>
    <tr><th>N°</th><th>Chantier</th><th>Client</th><th>Fin prévue</th><th>Statut</th></tr>
  </thead>
  <tbody>
    {% for c in chantiers %}
      <tr>
        <td>{{ c.id }}</td>
        <td>{{ c.nom }}</td>
        <td>{{ c.client }}</td>
        <td>{{ c.date_fin_prevue }}</td>
        <td class="statut-{{ c.statut }}">{{ c.statut }}</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
{% endblock %}
```

saisir.html

```

{% extends "base.html" %}
{% block titre %}Saisir une intervention{% endblock %}
{% block contenu %}
<h1>Saisir une intervention</h1>

{% if message %}<p class="succes">{{ message }}</p>{% endif %}

{% if erreurs %}
<ul class="erreurs">
    {% for e in erreurs %}<li>{{ e }}</li>{% endfor %}
</ul>
{% endif %}

<form method="post" action="{{ url_for('saisir') }}">
    <label for="chantier">Chantier</label>
    <select id="chantier" name="chantier">
        <option value="">-- choisir --</option>
        {% for c in chantiers %}
            <option value="{{ c.id }}" {% if saisie.get('chantier') == c.id|string %}
selected{% endif %}>{{ c.id }} · {{ c.nom }}</option>
        {% endfor %}
    </select>

    <label for="service">Service</label>
    <select id="service" name="service">
        <option value="">-- choisir --</option>
        {% for s in services %}
            <option value="{{ s }}" {% if saisie.get('service') == s %}selected{% endif
%}>{{ s }}</option>
        {% endfor %}
    </select>

    <label for="date">Date</label>
    <input id="date" name="date" type="text" placeholder="AAAA-MM-JJ"
value="{{ saisie.get('date', '') }}">

    <label for="duree">Durée en heures</label>
    <input id="duree" name="duree" type="text" placeholder="ex. 2.5"
value="{{ saisie.get('duree', '') }}">

    <label for="description">Ce qui a été fait</label>
    <textarea id="description" name="description"
rows="3">{{ saisie.get('description', '') }}</textarea>

    <label for="blocage">Blocage sur place (seulement s'il y en a eu un)</label>
    <input id="blocage" name="blocage" type="text" placeholder="ex. client absent"
value="{{ saisie.get('blocage', '') }}">

    <button type="submit">Enregistrer</button>
</form>
{% endblock %}

```

```
{% extends "base.html" %}
{% block titre %}Connexion{% endblock %}
{% block contenu %}
<h1>Connexion</h1>
{% if erreur %}<p class="erreur">{{ erreur }}</p>{% endif %}
<form method="post" action="{{ url_for('connexion') }}">
  <label for="login">Identifiant</label>
  <input id="login" name="login" type="text" autocomplete="username">
  <label for="mot_de_passe">Mot de passe</label>
  <input id="mot_de_passe" name="mot_de_passe" type="password"
autocomplete="current-password">
  <button type="submit">Se connecter</button>
</form>
{% endblock %}
```

A3 — Les données de TechnoVert et les comptes de test

Source : document de scénario, § 5, « Schéma relationnel de référence », pour les noms des colonnes et les trois services. § 5, « Nommage », pour les identifiants. Les valeurs des lignes sont écrites pour le module. Utilisée dans : LABO étapes 16 et 22, TP parties A à D.

Les données sont dans le fichier `chantiers.json`. Elles ont trois listes : les clients, les chantiers, les interventions. Les noms des clés sont ceux des colonnes de la base MySQL `technovert`.

Les clients

id	nom	ville	telephone	email
1	Commune de Vernay	Vernay	01 99 00 10 01	services.techniques@vernay.example
2	Golf des Trois Chênes	Montval	01 99 00 10 02	accueil@golf-trois-chenes.example
3	Copropriété Les Tilleuls	Vernay	01 99 00 10 03	syndic@les-tilleuls.example
4	Commune de Bellerive	Bellerive	01 99 00 10 04	mairie@bellerive.example

Les chantiers

Le statut vaut `ouvert` ou `closure`. On ne saisit plus rien sur un chantier clôturé.

id	client	nom	adresse	date_debut	date_fin_prevue	statut
1	Commune de Vernay	Arrosage du parc de la mairie	Parc de la mairie, Vernay	2026-03-02	2026-04-30	cloture
2	Golf des Trois Chênes	Arrosage des greens 1 à 9	Route de Montval, Montval	2026-04-06	2026-06-26	ouvert
3	Copropriété Les Tilleuls	Éclairage des allées	12 rue des Tilleuls, Vernay	2026-05-11	2026-06-12	ouvert
4	Commune de Bellerive	Éclairage du stade	Stade municipal, Bellerive	2026-06-01	2026-09-30	ouvert
5	Golf des Trois Chênes	Éclairage du parking	Route de Montval, Montval	2026-09-07	2026-11-27	ouvert
6	Commune de Vernay	Arrosage des ronds-points	Avenue de la Gare, Vernay	2026-09-14	2026-12-18	ouvert

Les interventions

Clé	Ce qu'elle contient
id	le numéro de l'intervention
id_chantier	le numéro du chantier
technicien	l'identifiant de la personne, au format pnom
service	atelier , bureau etudes ou commercial
date_intervention	la date, au format AAAA-MM-JJ
duree_heures	la durée, en heures, nombre décimal
description	ce qui a été fait
cloturee	true si l'intervention est terminée
blocage	un texte court s'il y a eu un empêchement sur place, sinon vide

id	chantier	technicien	service	date	durée (h)	description	clôturée	blocage
1	1	ngarcia	atelier	2026-03-03	6,0	Pose des programmeurs	oui	vide
2	1	ngarcia	atelier	2026-03-10	7,5	Raccordement des vannes	oui	vide
3	1	yropars	commercial	2026-04-28	1,0	Visite de réception avec le client	oui	vide
4	2	lmarchetti	bureau etudes	2026-04-02	3,0	Relevé des pentes des greens	oui	vide
5	2	lmoreau	atelier	2026-04-07	8,0	Tranchées et pose des tuyaux	oui	vide
6	2	lmoreau	atelier	2026-05-19	4,0	Réglage des arroseurs	non	acces refuse
7	3	ngarcia	atelier	2026-05-12	5,5	Pose des bornes lumineuses	non	vide
8	3	yropars	commercial	2026-05-20	0,5	Point d'avancement avec le syndic	non	client absent
9	4	lmarchetti	bureau etudes	2026-06-02	2,5	Calcul de l'éclairage du terrain	oui	vide
10	4	lmoreau	atelier	2026-06-15	8,0	Montage des mâts	non	vide
11	5	yropars	commercial	2026-09-08	1,0	Signature du devis sur place	non	vide
12	6	ngarcia	atelier	2026-09-15	6,0	Pose des goutteurs	non	vide

Les comptes de test

Tous les comptes ont le même mot de passe de test : **Chantier2026** . Le fichier ne contient pas les mots de passe, seulement leur empreinte.

Identifiant	Personne	Service
kbenali	Karim Benali, responsable d'atelier	atelier

Identifiant	Personne	Service
ngarcia	Nicolas Garcia, technicien	atelier
lmoreau	Laura Moreau, technicienne	atelier
lmarchetti	Ludivine Marchetti, chargée d'études	bureau etudes
yropars	Yann Ropars, commercial	commercial

A4 — Extrait de la documentation de Flask

Source : documentation officielle de Flask, version 3, pages « Quickstart » et « API », et de Jinja, page « Templates ». Extrait traduit et raccourci pour le module. Utilisée dans : LABO étape 8, TP parties B et C.

Les données de la requête. Dans une vue, l'objet `request` décrit la requête reçue.

- `request.method` : la méthode, "GET" ou "POST".
- `request.form` : les champs envoyés dans le corps de la requête, par un formulaire en POST.
- `request.args` : les paramètres écrits dans l'adresse, après le `?`.
- Pour lire un champ, on écrit `request.form["nom"]`. Si le champ manque, Flask répond avec l'erreur 400.
- Pour éviter cette erreur, on écrit `request.args.get("nom")`. On obtient `None` si le champ manque.

Les parties variables de l'adresse. Une partie de l'adresse entre chevrons devient un paramètre de la vue. Avec le convertisseur `int`, Flask n'accepte que des chiffres et donne un entier.

```
@app.route("/client/<int:numero>")
def fiche_client(numero):
    return f"Client numéro {numero}"
```

Les gabarits. `render_template("nom.html", cle=valeur)` remplit le gabarit avec les valeurs données. Dans le gabarit, `{{ cle }}` affiche la valeur.

Les conditions dans un gabarit. `{% if ... %}` choisit ce qui s'affiche. `or` donne un texte de secours quand la valeur est vide ou vaut `None`. Dans un menu, `selected` garde l'option choisie.

```
{% if service %}Service : {{ service }}{% else %}Tous les services{% endif %}
<option value="{{ s }}" {% if s == service %}selected{% endif %}>{{ s }}</option>
{{ i.blocage or "aucun" }}
```

Les adresses. `url_for("nom_de_la_vue")` renvoie l'adresse d'une vue. On lui passe les parties variables : `url_for("fiche_client", numero=4)`.

Les réponses spéciales.

- `redirect(adresse)` renvoie une redirection, code 302.
- `abort(404)` arrête la vue et renvoie la page « Not Found », code 404.

La session. `session` se lit et s'écrit comme un dictionnaire. Flask la range dans un cookie signé avec `app.secret_key`.

L'utilisateur peut lire le cookie. Il ne peut pas le modifier sans rendre la signature fausse.

- `session["cle"] = valeur` écrit une valeur.
- `"cle" in session` dit si la valeur existe.
- `session.clear()` vide la session.

Le mode développement. `flask run --debug` active le débogueur et le rechargement automatique. La documentation précise : n'utilisez jamais le mode développement sur un serveur de production.

A5 — Le point de départ du TP

↓ Télécharger [A5_consultation.zip](#)

Source : application écrite pour le module. Données : voir l'[annexe A3](#). Fichier [A5_consultation.zip](#) à télécharger ci-dessus. Utilisée dans : TP parties A à D.

Ce que contient le dossier

```
consultation/
  app.py           presque vide : vous écrivez les routes et les vues
  donnees.py       les fonctions qui lisent les données, décrites ci-
dessous
  chantiers.json    les données de l'annexe A3
  templates/base.html le cadre commun des pages : fourni
  templates/connexion.html la page de connexion : fournie
  static/style.css   la feuille de style : fournie
```

Vous créez vous-même `templates/liste.html` et `templates/detail.html`.

Le fichier app.py de départ

```
"""Interface de consultation des chantiers de TechnoVert.

Point de départ du TP. Écrivez ici les routes et les vues.
Noms de fonctions imposés : liste et detail. Les vues connexion et deconnexion sont
en bas.
"""

from flask import Flask, abort, redirect, render_template, request, session,
url_for

import donnees

app = Flask(__name__)
app.secret_key = "cle-de-tp-a-changer" # clé qui signe le cookie de session

# À compléter en partie D. Ces deux vues existent déjà,
# pour que les liens du menu de base.html marchent dès la partie A.
@app.route("/connexion", methods=["GET", "POST"])
def connexion():
    return render_template("connexion.html", erreur=None)

@app.route("/deconnexion")
def deconnexion():
    return redirect(url_for("connexion"))
```

Ce qu'attendent les gabarits fournis

- `base.html` contient les liens du menu. Il appelle `url_for('liste')`, `url_for('connexion')` et `url_for('deconnexion')`. Votre vue de liste doit donc s'appeler `liste`. Les vues `connexion` et `deconnexion` doivent rester.

- `connexion.html` attend une seule donnée : `erreur`, un texte ou `None`. Son formulaire envoie en POST les champs `login` et `mot_de_passe`.

Les fonctions de donnees.py

Fonction	Ce qu'elle renvoie
<code>donnees.lire_chantiers()</code>	La liste des chantiers, triés par date de début. Clés : <code>id</code> , <code>id_client</code> , <code>client</code> , <code>nom</code> , <code>adresse</code> , <code>date_debut</code> , <code>date_fin_prevue</code> , <code>statut</code> .
<code>donnees.lire_chantiers(service)</code>	Seulement les chantiers où ce service est intervenu.
<code>donnees.lire_chantier(numero)</code>	Un chantier, ou <code>None</code> s'il n'existe pas.
<code>donnees.lire_interventions(id_chantier=numero)</code>	Les interventions d'un chantier, triées par date.
<code>donnees.verifier_compte(login, mot_de_passe)</code>	<code>True</code> si le compte existe et si le mot de passe est juste.
<code>donnees.SERVICES</code>	<code>["atelier", "bureau etudes", "commercial"]</code>

Lancer le serveur

Ouvrez le dossier `consultation` dans VS Code, puis un terminal. Tapez

`python -m flask --app app run --debug`. Puis ouvrez `http://127.0.0.1:5000/chantiers` dans le navigateur.

Tant que la vue `liste()` n'est pas écrite, une erreur s'affiche : c'est normal. `/chantiers` affiche « Not Found ». `/connexion` et `/` affichent `BuildError` sur `liste`, car ils font un lien vers elle. Tout s'affiche dès que `liste()` existe.

Pour installer le dossier : téléchargez [A5_consultation.zip](#), puis clic droit, « Extraire tout », et avec « Parcourir », choisissez « Documents ».

A6 — Les écrans attendus

Source : demande de Karim Benali et de Yann Ropars, écrite pour le module. Utilisée dans : TP parties A à D.

Écran 1 — La liste des chantiers

- **Adresse** : `/chantiers`. L'adresse `/` y mène aussi.
- **Titre** : « Les chantiers ».
- **Filtre** : un menu « Service intervenu », avec « Tous les services » puis les trois services, et un bouton « Filtrer ».
- **Le choix du filtre** passe dans l'adresse : `/chantiers?service=atelier`. Après le filtre, le menu garde le service choisi.
- **Phrase de compte** : « 6 chantier(s). », ou « 5 chantier(s) pour le service atelier. ».
- **Tableau**, dans cet ordre : N°, Chantier, Client, Fin prévue, Statut. Le nom du chantier est un lien vers sa page.
- **Liste vide** : la phrase « Aucun chantier pour ce service. » à la place du tableau.

Écran 2 — Le détail d'un chantier

- **Adresse** : `/chantiers/` suivi du numéro, par exemple `/chantiers/2`.
- **En haut** : un lien « Retour à la liste ».
- **Titre** : le nom du chantier.
- **Informations** : client, adresse, début, fin prévue, statut.
- **Sous-titre** : « Interventions ».
- **Tableau**, dans cet ordre : Date, Technicien, Service, Durée (h), Ce qui a été fait, Blocage. Un blocage vide s'affiche « aucun ».
- **Aucune intervention** : la phrase « Aucune intervention sur ce chantier. ».
- **Numéro inconnu** : la page « Not Found », avec le code 404.

Écran 3 — La connexion

- **Adresse** : `/connexion`. Le gabarit est fourni.
- **Connexion réussie** : redirection vers la liste.
- **Échec** : la page revient, avec « Identifiant ou mot de passe incorrect. ».
- **Déconnexion** : `/deconnexion` vide la session et renvoie vers la connexion.

Règle d'accès

La liste et le détail sont réservés aux salariés connectés. Sans connexion, les deux adresses renvoient vers `/connexion`.

A7 — La charte de lisibilité minimale

Source : charte écrite pour le module, à partir des règles de présentation et de typographie à l'écran. Utilisée dans : TP partie D.

Une page de TechnoVert est lue vite, souvent sur un téléphone, par des gens pressés. Elle respecte ces huit règles.

1. **Un seul titre principal** par page, en `<h1>`. Il dit où l'on est : « Les chantiers », ou le nom du chantier.
2. **Une taille de texte d'au moins 16 pixels.** La feuille de style fournie règle 18 pixels : ne la réduisez pas.
3. **Du texte foncé sur un fond clair.** Pas de texte gris clair, pas de texte sur une image.
4. **Des tableaux avec une ligne d'en-tête** en `<th>`. Les colonnes sont dans l'ordre de l'écran. Une colonne de nombres dit son unité dans l'en-tête : « Durée (h) ».
5. **Aucune valeur brute d'informatique à l'écran** : ni `None`, ni `True`, ni `False`. Un statut s'écrit en français correct : « ouvert » ou « clôturé ».
6. **Des liens qui disent où ils mènent** : « Retour à la liste », le nom du chantier. Jamais « cliquez ici ».
7. **Une étiquette pour chaque champ de formulaire**, reliée au champ par `<label for="...">`.
8. **Un message quand une liste est vide**, à la place d'un tableau vide.