

Développement web : Python, HTML/CSS et base MySQL

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

U6-SP2-M3

module

Élève

document

TechnoVert

scénario 2

D recomposé · 6**h 30**

format et durée

Ce qu'il faut savoir avant de commencer	5
C'est quoi une fonction en Python ?	5
C'est quoi un dictionnaire ?	5
Comment lire un message d'erreur Python ?	5
Comment joindre un programme sur le réseau ?	6
C'est quoi une requête SQL ?	6
C'est quoi une clé étrangère ?	6
Quelques outils Python du labo	7
Comment lancer MySQL sur votre poste ?	7
Vérifiez-vous	8
Ce que fait le serveur, ce que fait le navigateur	9
Capsule 1 sur 5 Qui fait quoi : le serveur ou le navigateur	9
Capsule 2 sur 5 Route, vue, gabarit : le chemin d'une page	11
Capsule 3 sur 5 Recevoir une saisie et la vérifier	13
Capsule 4 sur 5 Se souvenir de l'utilisateur : la session	15
Capsule 5 sur 5 Quand le serveur plante	17
Ce qu'on retient	18
Ce que fait le serveur, ce que fait le navigateur	19
L'essentiel	19
Les mots à connaître	19
La méthode : écrire une vue qui reçoit un formulaire	20
Un exemple concret	20
Les pièges	20
Un mini-site à trois pages	22
PARTIE A — Afficher la liste	23
PARTIE B — Recevoir un formulaire	27
PARTIE C — Vérifier et protéger	32
PARTIE D — Lire une erreur serveur	38
POUR ALLER PLUS LOIN	41
CE QU'IL FAUT RETENIR	42

Brancher l'application sur la base	43
Capsule 1 sur 5 Parler à la base depuis Python	43
Capsule 2 sur 5 L'injection SQL et la requête paramétrée	45
Capsule 3 sur 5 Une couche d'accès aux données	48
Capsule 4 sur 5 Les identifiants hors du code	50
Capsule 5 sur 5 Un jeu de données qu'on peut rejouer	51
Ce qu'on retient	52
Brancher l'application sur la base	53
L'essentiel	53
Les mots à connaître	53
La méthode : écrire une fonction de la couche d'accès	54
Un exemple concret	54
Les pièges	55
Injection, correction, restauration	56
PARTIE A — Brancher l'application	57
PARTIE B — Isoler et sortir les secrets	63
PARTIE C — Injecter puis corriger	65
PARTIE D — Exporter et restaurer	69
POUR ALLER PLUS LOIN	72
CE QU'IL FAUT RETENIR	73
Un tableau de bord qui tient	74
COMMENT VOUS SEREZ NOTÉ	74
CE QU'ON VOUS DEMANDE	76
PAR OÙ COMMENCER	76
CE QUI DOIT MARCHER À LA FIN	77
Tous les documents du module	78
A1 — Le mini-site de départ et son mode d'emploi	78
A2 — Les gabarits du mini-site	81
A3 — Les données de TechnoVert et les comptes de test	86
A4 — Extrait de la documentation de Flask	89
A5 — L'application de départ et son mode d'emploi	90
A6 — Le jeu de données de la base	94

A7 — Le compte MySQL de l'application	98
A8 — Le fichier d'environnement à remplir	98
A9 — Extrait de la documentation du connecteur MySQL	99
A10 — Le script de vérification de la base	100
A11 — Le tableau de bord de départ et son mode d'emploi	102

Ce qu'il faut savoir avant de commencer

Ce qu'il faut savoir avant de commencer. Lisez-le si vous avez un doute. Environ 15 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.

C'est quoi une requête SQL ?

Une base de données range les données dans des tables. Chaque table a des colonnes et des lignes, comme un tableur.

On parle à la base en SQL. `SELECT` lit des lignes, `WHERE` les filtre, `INSERT` en ajoute.

```
SELECT nom, statut FROM chantier WHERE statut = 'ouvert';
INSERT INTO client (nom, ville) VALUES ('Mairie de Lusse', 'Lusse');
```

En SQL, un texte s'écrit entre apostrophes : `'ouvert'`. Un nombre s'écrit sans apostrophes.

C'est quoi une clé étrangère ?

Une colonne qui contient le numéro d'une ligne d'une autre table. Dans la table `chantier`, la colonne `id_client` contient le numéro du client.

De la même façon, dans la table `intervention`, la colonne `id_chantier` contient le numéro du chantier. C'est elle qui relie une intervention à son chantier.

Pour afficher le nom du client avec le chantier, on relie les deux tables avec `JOIN` :

```
SELECT chantier.nom, client.nom FROM chantier JOIN client ON client.id =
chantier.id_client;
```

Quelques outils Python du labo

Les labos vous font 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 ».

...

un tuple : `(3, "atelier")`

des valeurs entre parenthèses. Pour une seule valeur, `(nom,)` avec une virgule.

Comment lancer MySQL sur votre poste ?

Sur votre poste, MySQL tourne grâce à XAMPP. On le pilote depuis le panneau de contrôle de XAMPP :

- **Start**, en face de « MySQL », lance la base. Le nom « MySQL » passe au vert.
- **Stop** l'arrête.

On parle ensuite à la base dans un terminal, avec la commande `mysql -u root`.

Vérifiez-vous

-
- ☐ 1. Avec `c = {"id": 6, "statut": "ouvert"}`, que renvoie `c["statut"]` ?
-
- ☐ 2. Une trace se termine par `NameError: name 'chantier' is not defined`. Quel est le problème ?
-
- ☐ 3. Écrivez la requête qui lit le nom de tous les chantiers clôturés.
-
- ☐ 4. Dans la table `intervention`, quelle colonne relie une intervention à son chantier ?
-
- ☐ 5. MySQL ne répond plus. Où vérifiez-vous qu'il est lancé ?
-

Réponses

1. Le texte `"ouvert"`.
2. Le code utilise une variable `chantier` qui n'a jamais reçu de valeur. Il faut chercher la ligne indiquée juste au-dessus.
3. `SELECT nom FROM chantier WHERE statut = 'cloture';`
4. `id_chantier`, qui contient le numéro du chantier.
5. Dans le panneau de contrôle de XAMPP : « MySQL » doit être en vert. Sinon, cliquez sur « Start ».

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.

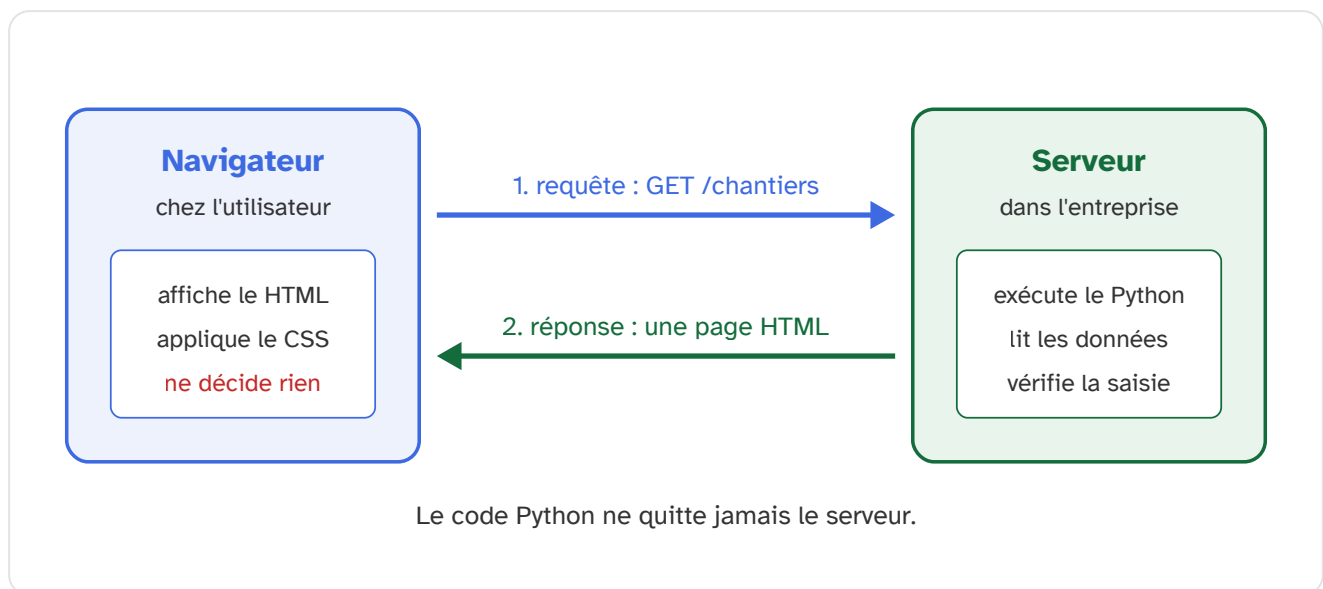


Schéma • Le navigateur demande, le serveur construit la page en Python et renvoie du HTML

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.

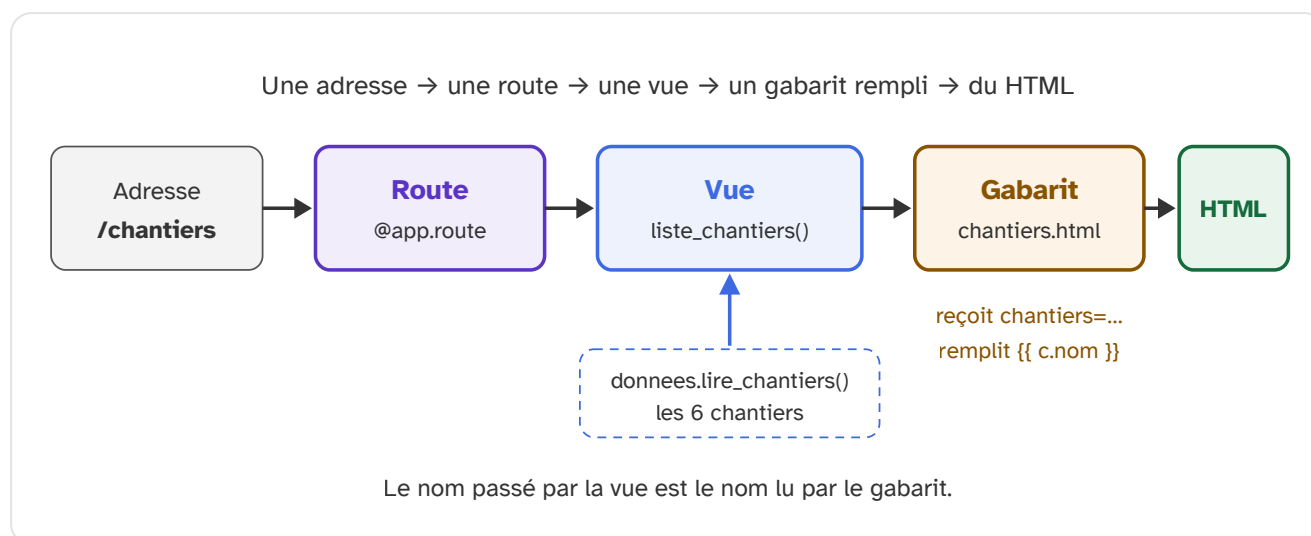


Schéma • De l'adresse /chantiers à la page affichée : route, vue, données, gabarit

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

- ✓ Après une connexion réussie, une redirection vers la liste

Pourquoi

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.

Brancher l'application sur la base

COURS-U6-S2-06

L'application de suivi des chantiers de TechnoVert marche. Les techniciens saisissent leurs interventions chaque soir. Mais les données sont dans un simple fichier, sur le serveur web.

Hélène Vasseur, la dirigeante, veut qu'elles passent dans la base MySQL de l'entreprise. Elle a une question : « Et si quelqu'un tape n'importe quoi dans une page, il peut abîmer la base ? »

La réponse est oui, si l'application est mal écrite. À la fin de la séance, vous saurez brancher l'application sur la base sans lui ouvrir cette porte.

LE PLAN DE LA SÉANCE

- 1 Parler à la base depuis Python.
- 2 L'injection SQL et la requête paramétrée.
- 3 Une couche d'accès aux données.
- 4 Les identifiants hors du code.
- 5 Un jeu de données qu'on peut rejouer.

LA QUESTION DE DÉPART

Sur un site, vous tapez une apostrophe dans un champ de recherche. La page affiche une erreur de base de données. Comment un seul caractère peut-il faire planter le site ?

CAPSULE 1 SUR 5

Parler à la base depuis Python

Commençons par le branchement lui-même. Comment un programme Python envoie-t-il une requête SQL à MySQL ?

Il passe par un **CONNECTEUR** : une bibliothèque Python qui sait parler à MySQL. Ici, c'est `mysql.connector`. Le connecteur ouvre une connexion, puis on lui confie un **CURSEUR** : l'objet qui envoie les requêtes et ramène les lignes.

Pensez à un appel téléphonique. On compose le numéro, on parle, on écoute la réponse, on raccroche.

Le connecteur, c'est le téléphone. Le curseur, c'est la conversation.

- 1 **Ouvrir** la connexion : l'adresse du serveur, le compte, le mot de passe, la base.
- 2 **Créer** un curseur.
- 3 **Exécuter** la requête avec `execute()`.
- 4 **Lire** les lignes avec `fetchall()`. Pour une écriture, **valider** avec `commit()`.
- 5 **Fermer** la connexion avec `close()`.

```
import mysql.connector

connexion = mysql.connector.connect(host="127.0.0.1", user="tv_appli",
                                   password="...", database="technovert")

curseur = connexion.cursor(dictionary=True)
curseur.execute("SELECT id, nom, statut FROM chantier")
chantiers = curseur.fetchall()      # une liste de dictionnaires
connexion.close()
```

Avec `dictionary=True`, chaque ligne arrive sous forme de dictionnaire : `{"id": 3, "nom": "Éclairage des allées", ...}`. C'est la forme qu'attendent les gabarits.

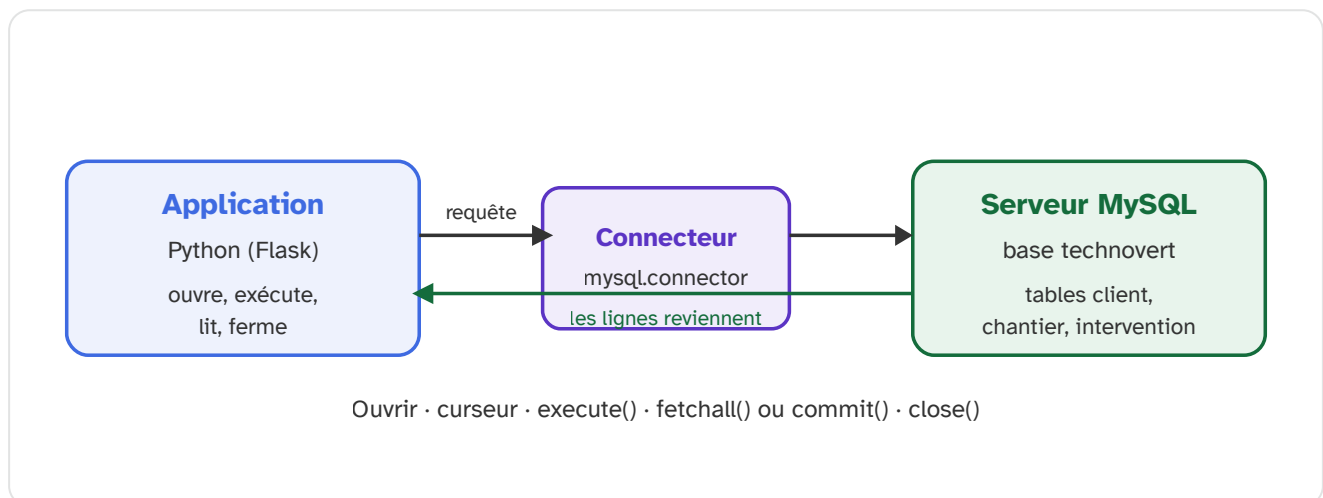


Schéma · L'application passe par le connecteur pour parler au serveur MySQL, puis ferme la connexion

ON LE FAIT ENSEMBLE — AJOUTER UNE INTERVENTION

1. Quelle est la première chose à faire ?
2. Quelle requête le curseur exécute-t-il ?
3. Faut-il `fetchall()` ?
4. Que faut-il ajouter avant de fermer ?

Exemple

Pourquoi

✓	Ouvrir, exécuter, lire, fermer, dans la même fonction	La connexion ne reste pas ouverte pour rien.
✗	Un <code>INSERT</code> sans <code>commit()</code>	Aucune erreur ne s'affiche, mais la ligne n'est jamais enregistrée.
✗	Oublier <code>close()</code> dans une page très visitée	Les connexions s'accumulent. MySQL finit par refuser les nouvelles.

CHECKPOINT

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

1. Quel objet envoie la requête et ramène les lignes ?
2. Un `UPDATE` qui clôture un chantier marche, mais le changement disparaît. Qu'est-ce qui manque ?

CAPSULE 2 SUR 5

L'injection SQL et la requête paramétrée

On sait maintenant envoyer une requête. Souvent, elle contient une valeur tapée par l'utilisateur : un service, un nom. Comment l'y mettre ?

La mauvaise façon, c'est de coller la saisie dans le texte de la requête. L'utilisateur peut alors taper un morceau de SQL. C'est l'**INJECTION SQL** : la saisie devient une partie de l'ordre donné à la base.

C'est comme un bon de commande où le client remplit la case « quantité ». S'il écrit « 2, et offrez-moi le reste du magasin », et que personne ne relit, le magasinier exécute tout.

- 1 Le code **colle** la saisie dans la requête, entre apostrophes.
- 2 L'utilisateur tape une **apostrophe** : elle ferme le texte trop tôt.
- 3 La suite de la saisie devient **du SQL**, que la base exécute.
- 4 La base fait ce que l'utilisateur a écrit, pas ce que le développeur voulait.

```
Code :      "... WHERE technicien = '" + login + "' AND service = '" + service +
            "'"

Saisie :    x' OR '1'='1

Requête :   ... WHERE technicien = 'ngarcia' AND service = 'x' OR '1'='1'
```

'1'='1' est toujours vrai. La base renvoie les interventions de **tous** les techniciens.

La parade s'appelle la **REQUÊTE PARAMÉTRÉE**. On écrit `%s` à la place de chaque valeur.

On donne les valeurs à part, dans un tuple. Le connecteur les envoie comme des données, jamais comme du SQL.

```
curseur.execute("SELECT ... WHERE technicien = %s AND service = %s", (login,
service))
```

Avec la même saisie, la base cherche un service qui s'appelle vraiment `x' OR '1'='1'`. Elle n'en trouve aucun.

Saisie collée : elle devient du SQL

```
"... WHERE service = '" + service + "'"
```

```
service = x' OR '1'='1
```



```
... WHERE service = 'x' OR '1'='1'
```

'1'='1' est toujours vrai → la base renvoie tout

Saisie en paramètre : elle reste une donnée

```
execute("... WHERE service = %s", (service,))
```

```
service = x' OR '1'='1
```

→ cherché tel quel, aucune ligne

Schéma • La saisie collée devient du SQL ; la saisie passée en paramètre reste une donnée

ON LE FAIT ENSEMBLE — LA RECHERCHE D'UN CHANTIER PAR NOM

1. Le code écrit `"... WHERE nom = ' " + nom + " ' "`. Est-il vulnérable ?
2. Par quoi remplace-t-on `' " + nom + " ' "` ?
3. Comment donne-t-on la valeur ?
4. Pourquoi une virgule dans `(nom, )` ?

Exemple

Pourquoi

✓	<code>execute("... WHERE id = %s", (numero,))</code>	La valeur part à part : elle ne peut pas changer la requête.
✗	<code>execute(f"... WHERE service = '{service}'")</code>	Une f-string colle la saisie, exactement comme <code>+</code> .
✗	<code>"... WHERE service = '%s'"</code> avec des apostrophes	Le connecteur ajoute les siennes. Le texte obtenu est faux.

INSTANT HISTOIRE — 2015 : TALKTALK ET SES VIEILLES PAGES

En 2015, TalkTalk est un grand opérateur de téléphone et d'Internet au Royaume-Uni.

Des pirates, dont des adolescents, trouvent d'anciennes pages de son site qui collent la saisie dans leurs requêtes.

Par injection SQL, ils volent les données d'environ 157 000 clients. L'entreprise reçoit une amende record de 400 000 livres.

Qu'est-ce qui aurait évité ça ?

CHECKPOINT

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

1. Réécrivez `"SELECT * FROM client WHERE ville = ' " + ville + " ' "` en requête paramétrée.
2. Un collègue dit : « Mon menu déroulant ne propose que trois services, pas de risque. » A-t-il raison ?

Une règle de droit, enfin. Tester une injection sur un site qui n'est pas le vôtre, sans autorisation écrite, est un délit. On ne s'entraîne que sur ses propres machines.

CAPSULE 3 SUR 5

Une couche d'accès aux données

La requête est sûre. Reste à savoir où ranger tout ce SQL. Dans chaque vue, au milieu du code de la page ?

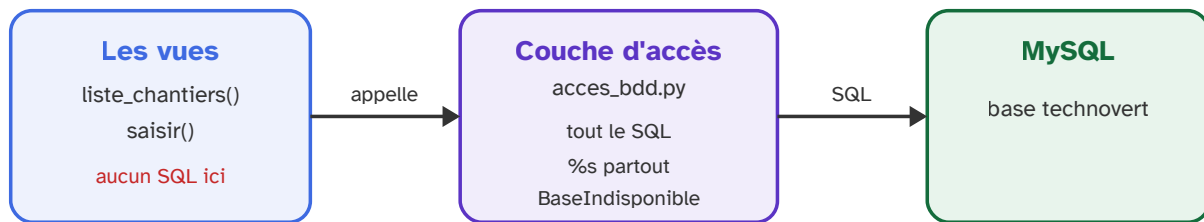
Non. On le range dans une **COUCHE D'ACCÈS AUX DONNÉES** : un module Python à part, qui contient toutes les requêtes.

Les vues appellent ses fonctions, comme `lire_chantiers()`. Elles ne voient jamais de SQL.

Au restaurant, le serveur ne va pas en cuisine. Il passe la commande au passe-plat, et le plat revient. Si la cuisine change de four, la salle ne voit aucune différence.

- 1 Toutes les requêtes vont dans **un seul fichier**, par exemple `acces_bdd.py`.
- 2 Chaque besoin devient **une fonction** au nom clair : `lire_chantier(numero)`.
- 3 Les vues **appellent** ces fonctions et reçoivent des listes ou des dictionnaires.
- 4 La couche **traduit les pannes** : si MySQL ne répond pas, elle lève une erreur à elle, `BaseIndisponible`.
- 5 L'application **répond proprement** à cette erreur : une page « Service indisponible », code 503.

Le code 503 veut dire : le serveur marche, mais il attend un autre service qui ne répond pas. L'utilisateur voit un message clair, pas une trace.



Pour changer de base, on touche la couche seule. Les vues ne bougent pas.

Schéma • Les vues appellent la couche d'accès ; seule la couche parle à MySQL

ON LE FAIT ENSEMBLE — LA BASE S'ARRÊTE PENDANT LA NUIT

1. Où l'erreur de connexion apparaît-elle en premier ?
2. Que fait `ouvrir()` de cette erreur ?
3. Que voit le technicien qui ouvre la liste ?
4. Où le technicien de maintenance lit-il la vraie cause ?

Exemple

Pourquoi

✓	On passe du fichier à MySQL en changeant une ligne d'import	Les vues n'ont pas bougé : c'est la preuve que la couche est bien isolée.
✗	Un <code>SELECT</code> écrit dans la vue <code>liste_chantiers()</code>	Le SQL se disperse. Pour vérifier les injections, il faut relire tout le site.
✗	La trace MySQL affichée à l'utilisateur quand la base tombe	Elle montre le nom du serveur et du compte. Et l'utilisateur ne sait pas quoi faire.

CHECKPOINT

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

1. Vous cherchez toutes les requêtes de l'application pour les vérifier. Où regardez-vous ?
2. Que veut dire le code 503 ?

Les identifiants hors du code

La couche d'accès se connecte avec un compte et un mot de passe. Où les écrire ?

Surtout pas dans le code. Le code se copie, s'envoie, se range dans un dépôt partagé. Le mot de passe partirait avec.

On le met dans une **VARIABLE D'ENVIRONNEMENT** : une valeur que le système donne au programme quand il le lance.

C'est comme le code du portail d'une résidence. On ne le grave pas sur le portail. Chaque habitant le connaît, et on le change sans changer le portail.

- 1 On écrit les identifiants dans un **fichier d'environnement**, hors du dossier du code.
- 2 On le **protège** : on le range dans son dossier personnel, que les autres comptes de la machine ne lisent pas. Sur un serveur Linux, on ajoute `chmod 600`.
- 3 L'application le **charge** au démarrage, avec `load_dotenv()`.
- 4 Le code les lit avec `os.environ`.

```
Fichier technover.env (dossier perso)  Code de acces_bdd.py
TV_BDD_UTILISATEUR=tv_appli             user=os.environ["TV_BDD_UTILISATEUR"]
TV_BDD_MOT_DE_PASSE=...                 password=os.environ["TV_BDD_MOT_DE_PASSE"]
```

Autre gain : le même code tourne sur votre poste de test et sur le serveur de production. Seul le fichier d'environnement change.

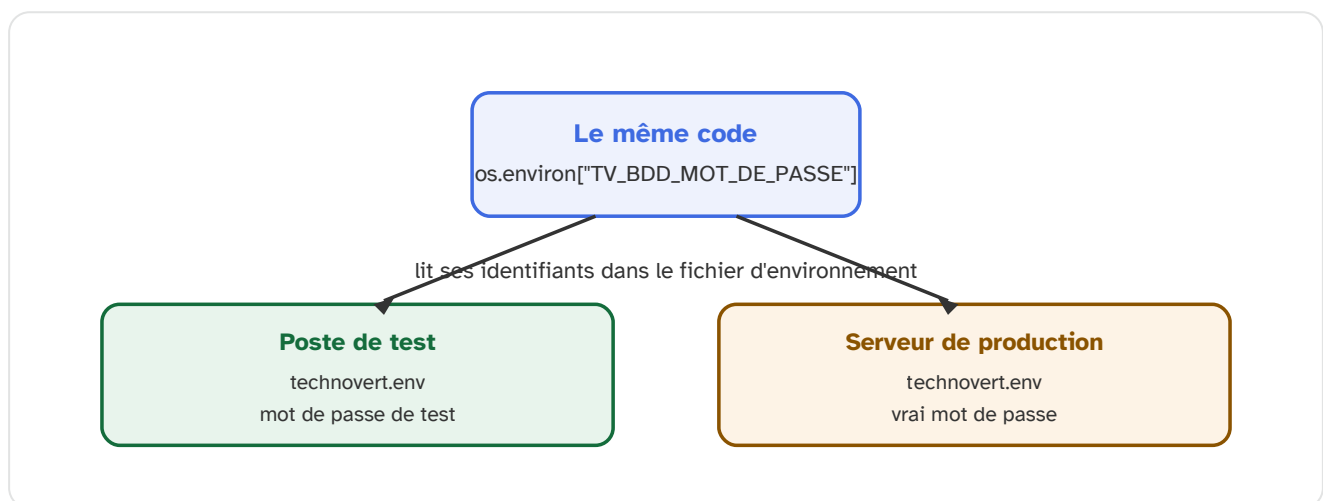


Schéma • Le code reste le même ; seul le fichier d'environnement change entre le test et la production

ON LE FAIT ENSEMBLE — LA CLÉ SECRÈTE DE LA SESSION

1. Le code contient `app.secret_key = "cle-de-labo"` . Est-ce un secret ?
2. Où la mettre ?
3. Qu'écrit-on dans le code ?

Exemple

Pourquoi

- | | | |
|---|---|--|
| ✓ | Un fichier d'exemple livré avec le code, avec <code>a-remplacer</code> comme valeur | Il montre les noms attendus, sans aucun vrai secret. |
| ✗ | Le fichier d'environnement rangé dans le dossier du code | Il part dans le ZIP ou dans le dépôt, avec le code. |
| ✗ | Le mot de passe écrit en commentaire, « pour mémoire » | Un commentaire se lit aussi bien qu'une ligne de code. |

CHECKPOINT

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

1. On passe de votre poste de test au serveur de production. Que modifie-t-on ?
2. Sur le serveur Linux de production, que fait `chmod 600` sur le fichier d'environnement ?

CAPSULE 5 SUR 5

Un jeu de données qu'on peut rejouer

Dernier outil, pour tester sans crainte. Pendant un test, on ajoute des lignes, on en casse. Comment revenir à un état connu ?

On garde un **jeu de données de test** : un script SQL qui efface la base et la recrée, toujours à l'identique. On le rejoue avant chaque série de tests. Un script de vérification compte ensuite les lignes et teste les cas connus.

```
mysql -u root -e "source technover.sql"    # la base revient à l'état connu
python verifier_base.py                    # chaque ligne doit dire OK
```

Le même couple sert à prouver une restauration. On restaure la base sur une machine neuve, puis on lance la vérification. Si tout est OK, la restauration a marché.

Ce qu'on retient

SI VOUS NE DEVEZ RETENIR QUE TROIS CHOSES

1. Une valeur saisie ne se colle jamais dans une requête : on écrit `%s` et on passe la valeur à part.
2. Tout le SQL vit dans la couche d'accès. Les vues appellent des fonctions, et une base arrêtée donne une page 503 propre.
3. Les identifiants vivent dans un fichier d'environnement protégé, jamais dans le code.

Revenons à Hélène Vasseur. Oui, quelqu'un peut taper n'importe quoi dans une page.

Avec des requêtes paramétrées, sa saisie reste une simple donnée, et la base ne l'exécute pas. Et le mot de passe de la base n'est écrit nulle part dans le code.

Brancher l'application sur la base

COURS-U6-S2-06

L'essentiel

- **Connecteur** : `mysql.connector` relie Python à MySQL. Ouvrir, curseur, `execute()`, `fetchall()` ou `commit()`, `close()`.
- **Injection SQL** : une saisie collée dans la requête devient du SQL.
- **Requête paramétrée** : `%s` dans la requête, les valeurs à part dans un tuple.
- **Couche d'accès** : tout le SQL dans un seul module. Les vues appellent ses fonctions.
- **Base indisponible** : la couche lève `BaseIndisponible`, l'application répond 503 avec une page claire.
- **Identifiants** : dans un fichier d'environnement protégé, lus avec `os.environ`.
- **Jeu de test** : un script SQL rejouable, puis un script de vérification.

Les mots à connaître

Mot	En clair
Connecteur	La bibliothèque qui permet à Python de parler à MySQL.
Curseur	L'objet qui envoie une requête et ramène les lignes.
Injection SQL	Une saisie qui change la requête envoyée à la base.
Requête paramétrée	Une requête avec <code>%s</code> , dont les valeurs sont envoyées à part.
Couche d'accès	Le module qui contient toutes les requêtes de l'application.
Variable d'environnement	Une valeur donnée au programme par le système, hors du code.
Bibliothèque, module	Un fichier de code déjà écrit qu'on réutilise. Le connecteur est une bibliothèque ; la couche d'accès est un module.

<code>127.0.0.1</code>	L'adresse de la machine elle-même. La base tourne sur la même machine que l'application.
<code>chmod 600</code>	Sur un serveur Linux, rend un fichier lisible par son seul propriétaire. On l'utilise pour le fichier d'environnement.
f-string	Un texte Python écrit <code>f"...{x}..."</code> , où <code>{x}</code> est remplacé par la valeur. À ne jamais utiliser pour une requête SQL.

La méthode : écrire une fonction de la couche d'accès

1. **Ouvrez** la connexion avec `ouvrir()`.
2. **Écrivez** la requête avec un `%s` par valeur, sans apostrophes autour.
3. **Exécutez** avec les valeurs dans un tuple : `(valeur,)`.
4. **Lisez** avec `fetchall()`, ou **validez** avec `commit()`.
5. **Fermez** la connexion, puis renvoyez le résultat.

Un exemple concret

Lire les interventions d'un service, sans risque d'injection.

```
def interventions_du_service(service):
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute("SELECT * FROM intervention WHERE service = %s", (service,))
    lignes = curseur.fetchall()
    connexion.close()
    return lignes
```

Saisie	Ce que la base reçoit	Résultat
<code>atelier</code>	le service <code>atelier</code>	les interventions de l'atelier
<code>x' OR '1'='1</code>	un service nommé <code>x' OR '1'='1</code>	aucune ligne

Conclusion : la saisie piégée est traitée comme une simple donnée.

Les pièges

- **✗ Une f-string dans `execute()`** → toujours `%s` et un tuple.
- **✗ `'%s'` entre apostrophes** → `%s` seul.
- **✗ Oublier `commit()`** après une écriture → l'écriture est perdue.
- **✗ Le mot de passe dans le code** → fichier d'environnement, rangé dans votre dossier personnel.

Injection, correction, restauration

LABO-U6-S2-06

Contexte — Vous reprenez l'application de suivi des chantiers de TechnoVert. Elle marche, mais elle range ses données dans un fichier. Hélène Vasseur veut qu'elles passent dans la base MySQL de l'entreprise.

Un collègue a repris une vieille page écrite par l'ancien prestataire. Elle colle la saisie dans sa requête. Vous allez le prouver, puis corriger.

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 brancher une application sur MySQL, isoler l'accès aux données, corriger une injection SQL et prouver une restauration.

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
Connecteur	La bibliothèque <code>mysql.connector</code> qui relie Python à MySQL.
Curseur	L'objet qui envoie une requête et ramène les lignes.
Injection SQL	Une saisie qui change la requête envoyée à la base.
Requête paramétrée	Une requête avec <code>%s</code> , dont les valeurs sont passées à part.
Couche d'accès	Le module <code>acces_bdd.py</code> qui contient toutes les requêtes.
Restaurer	Recréer la base à l'identique à partir d'un fichier d'export.

Ce dont vous avez besoin

- Votre poste Windows, avec XAMPP, Python, VS Code et un navigateur.
- [Annexe A5](#) : l'application de départ et son mode d'emploi. Le dossier se télécharge dans [A5_technovert_web.zip](#).
- [Annexe A6](#) : le jeu de données de la base, fichier [A6_technovert.sql](#).
- [Annexe A7](#) : le compte MySQL de l'application, fichier [A7_compte_appli.sql](#).
- [Annexe A8](#) : le fichier d'environnement à remplir.
- [Annexe A9](#) : un extrait de la documentation du connecteur MySQL.
- [Annexe A10](#) : le script de vérification, fichier [A10_verifier_base.py](#).

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 — Brancher l'application	1 à 6	l'application qui lit la base MySQL	35 min
B — Isoler et sortir les secrets	7 à 11	une couche d'accès propre, sans mot de passe dans le code	25 min
C — Injecter puis corriger	12 à 17	une injection prouvée, puis rendue inoffensive	35 min
D — Exporter et restaurer	18 à 22	une base restaurée sur une pile vierge, vérifiée	25 min

PARTIE A — Brancher l'application

Étapes 1 à 6, environ 35 min. À la fin, l'application affiche les chantiers lus dans MySQL.

Besoin d'un rappel ? Revoir le cours : [« Parler à la base depuis Python »](#).

01 Installer l'application et lancer MySQL

- ☐ Téléchargez `A5_technovert_web.zip` depuis l'[annexe A5](#). Faites un clic droit dessus, « Extraire tout », puis avec « Parcourir », choisissez votre dossier « Documents ».
- ☐ Ouvrez le dossier `Documents\technovert-web` dans VS Code. Ouvrez un terminal : menu « Terminal », puis « Nouveau terminal ».
- ☐ Ouvrez le panneau de contrôle de XAMPP. Cliquez sur « Start », en face de « MySQL ».

Vous devez voir : « MySQL » en vert dans le panneau de XAMPP. Dans VS Code, le dossier contient `app.py`, `acces_bdd.py` et les fichiers [A6_technovert.sql](#), [A7_compte_appli.sql](#), [A10_verifier_base.py](#).

02 Créer la base et son compte

- ☐ Dans le terminal, vérifiez que la commande `mysql` répond :

```
mysql --version
```

Si vous voyez « mysql n'est pas reconnu » : suivez l'[annexe A5](#), « Rendre la commande mysql disponible », puis rouvrez VS Code.

- ☐ Chargez le jeu de données, puis le compte de l'application :

```
mysql -u root -e "source A6_technovert.sql"
mysql -u root -e "source A7_compte_appli.sql"
```

☐ Vérifiez que la base existe :

```
mysql -u root -e "SELECT COUNT(*) FROM technovert.chantier;"
```

Vous devez voir : aucun message pour les deux premières commandes, puis un petit tableau avec le nombre 6.

03 Écrire la connexion et la lecture des chantiers

La couche d'accès est le fichier `acces_bdd.py`. Ses premières lignes, déjà écrites, chargent vos identifiants avec `load_dotenv`. Deux fonctions sont à écrire.

☐ Dans `acces_bdd.py`, remplacez tout le corps de `ouvrir()`, `pass` compris, par :

```
try:
    return mysql.connector.connect(
        host=os.environ["TV_BDD_HOTE"],
        user=os.environ["TV_BDD_UTILISATEUR"],
        password=os.environ["TV_BDD_MOT_DE_PASSE"],
        database=os.environ["TV_BDD_BASE"],
        connection_timeout=3)
except mysql.connector.Error as erreur:
    raise BaseIndisponible(str(erreur)) from erreur
```

☐ Remplacez tout le corps de `lire_chantiers()`, `return []` compris, par :

```
connexion = ouvrir()
curseur = connexion.cursor(dictionary=True)
curseur.execute(COLONNES_CHANTIER + "ORDER BY ch.date_debut")
lignes = curseur.fetchall()
connexion.close()
return lignes
```

- ☐ Lisez la ligne `COLONNES_CHANTIER`, écrite en haut du fichier. Notez la table reliée à `chantier` par le `JOIN` : _____

Le `+` colle ici deux textes fixes, écrits par vous. Ce n'est pas une saisie d'utilisateur : il n'y a donc aucun risque d'injection.

Vous devez avoir : un fichier enregistré avec Ctrl+S, et le nom d'une table relié par le `JOIN`. On teste le tout à l'étape 5.

Pourquoi ? Pourquoi le `SELECT` relie-t-il deux tables au lieu de lire la seule table `chantier` ?

Votre réponse :

04 Remplir le fichier d'environnement

Le fichier d'environnement se range dans votre dossier personnel, jamais dans le dossier du code. L'application le lit seule au démarrage.

- ☐ Copiez le modèle de l'[annexe A8](#) dans votre dossier personnel :

```
Copy-Item technovert.env.exemple $HOME\technovert.env
```

- ☐ Ouvrez-le avec `notepad $HOME\technovert.env`. Mettez le mot de passe du compte `tv_appli` de l'[annexe A7](#), et une longue valeur au hasard pour `TV_CLE_SECRETE`. Enregistrez.

- ☐ Vérifiez son contenu :

```
Get-Content $HOME\technovert.env
```

Vous devez voir : quatre lignes de commentaire, puis vos cinq lignes `TV_...`, sans aucun `a-` remplacer. Les accents des commentaires peuvent mal s'afficher : ce n'est pas grave.

05 Basculer l'application sur la base

L'application importe encore l'ancienne couche, celle du fichier. On la fait pointer sur la base.

- ☐ Ouvrez `app.py`. Repérez la ligne `import donnees`.

- ☐ Remplacez-la par :

```
import acces_bdd as donnees
```

- ☐ Remplacez la ligne de la clé secrète par :

```
app.secret_key = os.environ["TV_CLE_SECRETE"]
```

- ☐ Dans `app.py`, juste au-dessus de `from flask import ...`, ajoutez la ligne `import os`. Enregistrez, puis lancez le serveur :

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

- ☐ Connectez-vous sur `/connexion` avec le compte `ngarcia`, mot de passe `Chantier2026`. Les autres logins sont dans l'[annexe A6](#), table `compte`. Le mot de passe de test est le même pour tous. Puis ouvrez `http://127.0.0.1:5000/chantiers`.

Vous devez voir : les six chantiers, comme avant. Cette fois, ils viennent de MySQL.

Si vous voyez la page « Service momentanément indisponible » : MySQL est arrêté dans XAMPP, ou le mot de passe de `technovert.env` est faux. Corrigez, puis rechargez la page.

06 Vérifier que les données viennent de la base

- ☐ Ouvrez un second terminal avec le bouton « + » du panneau Terminal. Le premier fait tourner le serveur.
- ☐ Changez un nom de chantier directement dans la base :

```
mysql -u root -e "UPDATE technovert.chantier SET nom='Essai base' WHERE id=1;"
```

- ☐ Rechargez `/chantiers` dans le navigateur.

Vous devez voir : le premier chantier s'appelle maintenant « Essai base ». L'application lit bien la base en direct.

POINT DE CONTRÔLE A

- ☐ Montrez au formateur la liste avec « Essai base », et votre réponse à l'étape 3.
- ☐ Le formateur a validé. Passez à la partie B.

PARTIE B — Isoler et sortir les secrets

Étapes 7 à 11, environ 25 min. À la fin, tout le SQL est dans la couche d'accès, et aucun mot de passe n'est dans le code.

Besoin d'un rappel ? Revoir le cours : « [Une couche d'accès aux données](#) ».

07 Vérifier que les vues ne contiennent pas de SQL

- ☐ Cherchez le mot `SELECT` dans le fichier des vues :

```
Select-String -Pattern "SELECT", "INSERT", "mysql" -Path app.py
```

Vous devez voir : aucune ligne. Tout le SQL est dans `acces_bdd.py`.

Pourquoi ? Pourquoi ranger toutes les requêtes dans un seul fichier, au lieu d'en mettre dans chaque vue ?

Votre réponse :

08 Provoquer une panne de base

☐ Laissez le serveur tourner. Dans le panneau de XAMPP, cliquez sur « Stop », en face de « MySQL ».

☐ Rechargez `/chantiers` dans le navigateur.

Vous devez voir : la page « Service momentanément indisponible », pas une trace d'erreur.

09 Lire la vraie cause dans le journal

☐ Regardez le terminal du serveur.

Vous devez voir : une ligne qui contient `ERROR in app: Base indisponible`, suivie de la cause. La ligne juste en dessous se termine par `503 -`.

☐ Dans le panneau de XAMPP, cliquez sur « Start » en face de « MySQL ». Rechargez la page.

Vous devez voir : la liste des chantiers revient.

10 Trouver dans la doc

☐ Ouvrez l'[annexe A9](#).

☐

Trouvez comment passer une valeur à une requête sans la coller dans le texte. Notez le symbole employé dans la requête : _____

☐ Notez sous quelle forme les valeurs sont passées à `execute()` : _____

Vous devez avoir : un symbole, et le mot « tuple ». Ils servent en partie C.

11 Vérifier qu'aucun secret n'est dans le code

☐ Cherchez un mot de passe en clair dans les fichiers du code :

```
Select-String -Pattern "Labo-Appli", "password=" -Path app.py, acces_bdd.py
```

Vous devez voir : la seule ligne trouvée est `password=os.environ[...]`. Aucun mot de passe en clair.

POINT DE CONTRÔLE B

☐ Montrez au formateur la page 503, la ligne du journal, et vos réponses aux étapes 7 et 10.

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

PARTIE C — Injecter puis corriger

Étapes 12 à 17, environ 35 min. À la fin, une injection est prouvée, puis rendue inoffensive.

Besoin d'un rappel ? Revoir le cours : [« L'injection SQL et la requête paramétrée »](#).

Changez de rôle : le lecteur devient pilote.

Cadre légal. Vous testez une injection sur votre propre machine, dans un but d'apprentissage. Le faire sur un site qui n'est pas le vôtre, sans autorisation écrite, est un délit.

12 Lire la page vulnérable

Un collègue a repris la page « Mes interventions » de l'ancien prestataire. Sa fonction est `interventions_du_technicien`, en bas de `acces_bdd.py`.

☐ Ouvrez `acces_bdd.py` et lisez cette fonction.

☐ Notez comment le login et le service entrent dans la requête :

Vous devez avoir : les valeurs collées dans le texte avec des `+` et des apostrophes.

13 Voir la page normale

☐ Si le serveur est arrêté, relancez-le dans le premier terminal :

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

☐ Connectez-vous avec le compte `ngarcia`, mot de passe `Chantier2026`.

☐ Ouvrez `/mes-interventions?service=atelier`.

Vous devez voir : la phrase `4 intervention(s) pour le service « atelier ».`, et un tableau de 4 lignes, toutes de `ngarcia`.

14 Prouver l'injection

- ☐ Dans la barre d'adresse, remplacez le service par une injection. Tapez cette adresse, puis validez :

```
http://127.0.0.1:5000/mes-interventions?service=x' OR '1'='1
```

Vous devez voir : un nombre d'interventions bien plus grand que 4, et des interventions d'autres techniciens. Vous voyez des données qui ne sont pas les vôtres.

- ☐ Notez le nombre d'interventions affiché : _____
- ☐ Faites tout de suite une capture d'écran de cette page. C'est votre preuve « avant correction » : après l'étape 15, vous ne pourrez plus la refaire.

Pourquoi ? Pourquoi la base a-t-elle renvoyé les interventions de tout le monde ?

Votre réponse :

15 Corriger avec une requête paramétrée

- ☐ Dans `interventions_du_technicien`, remplacez les lignes de `requete = (` jusqu'à `curseur.execute(requete)` par :

```
requete = ("SELECT i.date_intervention, c.nom AS chantier, i.technicien,
i.service, "
          "i.duree_heures FROM intervention i JOIN chantier c ON c.id =
i.id_chantier "
          "WHERE i.technicien = %s AND i.service = %s "
          "ORDER BY i.date_intervention")
curseur.execute(requete, (login, service))
```

☐ Enregistrez le fichier.

Vous devez voir : le fichier s'enregistre sans erreur. On teste l'effet à l'étape 16.

16 Vérifier que l'attaque échoue

☐ Rechargez la page normale `/mes-interventions?service=atelier`.

Vous devez voir : les 4 interventions de `ngarcia`, comme avant. La correction n'a rien cassé.

☐ Rechargez l'adresse d'injection de l'étape 14.

Vous devez voir : « 0 intervention(s) ». La base a cherché un service qui s'appelle vraiment `x' OR '1'='1`. Il n'existe pas.

17 Garder la preuve

Vous avez déjà la capture « avant correction », faite à l'étape 14.

☐ Faites maintenant une capture de la même adresse d'injection après correction : « 0 intervention(s) ».

Vous devez avoir : deux captures, l'une de l'étape 14, l'autre d'ici. Vous les montrez au point de contrôle C.

POINT DE CONTRÔLE C

- ☐ Montrez au formateur les deux preuves et votre réponse à l'étape 14.
- ☐ Le formateur a validé. Passez à la partie D.

PARTIE D — Exporter et restaurer

Étapes 18 à 22, environ 25 min. À la fin, la base est restaurée sur une pile vierge, et vérifiée.

Besoin d'un rappel ? Revoir le cours : [« Un jeu de données qu'on peut rejouer »](#).

Le jeu de test de l'[annexe A6](#) recrée une base neuve, toujours la même. Un export, lui, sauvegarde la **vraie** base, avec les saisies des utilisateurs. C'est cet export qu'on restaure quand une machine tombe en panne. On le prouve ici sur une base neuve.

18 Exporter la base

- ☐ Exportez toute la base dans un fichier, dans le dossier de l'application :

```
mysqldump -u root technoverd --result-file=export_technoverd.sql
```

- ☐ Cherchez dans le fichier les lignes qui créent les tables et qui ajoutent les données :

```
Select-String -Pattern "CREATE TABLE", "INSERT INTO" -Path export_technoverd.sql
```

Vous devez voir : des lignes `CREATE TABLE` pour les quatre tables, et des lignes `INSERT INTO`. L'export contient la structure et les données.

19 Simuler une pile vierge

Une pile vierge, c'est une base qui n'existe pas encore. On efface la base pour le simuler.

-
- ☐ Supprimez la base :

```
mysql -u root -e "DROP DATABASE technover;"
```

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

Vous devez voir : la page « Service momentanément indisponible ». La base n'existe plus.

20 Restaurer

-
- ☐ Recréez la base vide, puis rechargez l'export dedans :

```
mysql -u root -e "CREATE DATABASE technover;"  
mysql -u root technover -e "source export_technover.sql"
```

-
- ☐ Rechargez `/chantiers`.

Vous devez voir : les six chantiers reviennent.

21 Vérifier la restauration avec le script

Une page qui s'affiche ne prouve pas que tout est là. Le script de l'[annexe A10](#) compte les lignes et teste les cas connus.

-
- ☐ Dans le second terminal, lancez le script de vérification. Il lit lui aussi `technovert.env` :

```
python A10_verifier_base.py
```

Vous devez voir : chaque ligne commence par OK, et la dernière ligne dit « 13 vérification(s) sur 13 réussie(s). ».

Si vous voyez un ÉCHEC : la restauration est incomplète, ou le jeu de test n'est pas le bon. Reprenez à l'étape 20.

22 Trouver dans la doc et conclure

-
- ☐ Ouvrez l'[annexe A10](#). Trouvez la dernière vérification du script, celle qui teste une injection.

- ☐ Notez ce qu'elle envoie comme service, et le résultat attendu :

Vous devez avoir : un service piégé, avec des apostrophes, et zéro ligne attendue.

Pourquoi ? Pourquoi vérifier une injection dans un script de recette, en plus de l'avoir corrigée ?

Votre réponse :

POINT DE CONTRÔLE D

☐ Montrez au formateur le script qui affiche 13 sur 13, et vos réponses aux étapes 22.

☐ 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.

Une IA a proposé cette fonction pour chercher un client par sa ville. Elle marche, mais elle contient trois défauts.

```
def clients_par_ville(ville):
    connexion = mysql.connector.connect(host="127.0.0.1", user="root",
                                       password="root",
                                       database="technovert")
    curseur = connexion.cursor(dictionary=True)
    curseur.execute(f"SELECT * FROM client WHERE ville = '{ville}'")
    return curseur.fetchall()
```

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

Ligne	Le défaut	Le danger

CE QU'IL FAUT RETENIR

1. Une valeur saisie ne se colle jamais dans une requête : on écrit `%s` et on passe la valeur à part.
2. Tout le SQL vit dans la couche d'accès. Une base arrêtée donne une page 503 propre, jamais une trace.
3. Une restauration ne se croit pas : elle se prouve avec un script de vérification.

Un tableau de bord qui tient

MISE EN SITUATION-U6-S2-M6

Contexte — Karim Benali veut un tableau de bord pour suivre les interventions de l'atelier. Il sera exposé aux utilisateurs de TechnoVert.

Hélène Vasseur a une exigence : « Je ne veux pas qu'un petit malin casse la base en tapant dans une case. » Le formateur jouera cet utilisateur malveillant.

RÈGLE IA DE LA SÉANCE

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

Si vous avez posé une question à une IA, dites-le pendant l'entretien. C'est pris en compte, pas reproché.

COMMENT VOUS SEREZ NOTÉ

Il n'y a aucun fichier à rendre. Vous êtes noté **à l'oral**, sur votre poste, pendant un entretien de 5 minutes avec le formateur. Il vient vous voir dès que votre tableau marche.

Pendant l'entretien, le formateur :

1. vous demande de montrer votre tableau, sans filtre puis avec un filtre ;
2. tape lui-même une saisie piégée dans l'adresse de votre page ;
3. vous pose des questions sur votre code ;
4. vous demande une petite modification, à faire devant lui en 2 minutes ;
5. regarde votre façon de travailler et d'expliquer.

Il suit cette grille. Lisez-la avant de commencer : c'est exactement ce qui sera noté.

Critère	Insuffisant (0 ou 1)	Fragile (2)	Acquis (3)	Maîtrisé (4)
Démonstration · le tableau lit la base	La page ne s'affiche pas, ou plante sur un cas normal.	La page s'affiche, mais le filtre ou le nom du chantier manque.	Toutes les interventions, avec leur chantier, et le filtre par service marche.	Tout marche, et le total d'heures correspond aux lignes montrées.
Résistance · la saisie piégée reste sans effet	La saisie affiche des données en plus, ou une trace d'erreur.	La saisie est bloquée, mais seulement par le menu ou par un test fragile.	La requête est paramétrée : la saisie ne change rien.	La requête est paramétrée, et vous le prouvez vous-même devant le formateur.
Questions techniques · vous expliquez votre code	Vous ne savez pas dire ce que fait votre requête.	Vous l'expliquez avec de l'aide.	Vous expliquez la requête, le <code>JOIN</code> et le rôle de <code>%s</code> .	Vous expliquez aussi, avec vos mots, pourquoi l'attaque échoue.
Travail personnel · la modification en direct	Vous ne trouvez pas où modifier.	Vous trouvez l'endroit, mais la modification n'aboutit pas.	La modification marche en 2 minutes.	Elle marche, et vous la testez vous-même sans qu'on vous le demande.
Professionnalisme · votre façon de travailler	Réponses floues, dossier en désordre, rien n'est dit de ce qui ne marche pas.	Vocabulaire approximatif, explications à reprendre.	Mots du métier, explications claires. Vous dites ce qui ne marche pas et l'aide reçue.	En plus, vous signalez un risque qui reste, ou une amélioration possible.

La note est la somme des cinq lignes, sur 20.

Ce dont vous disposez

- Votre poste Windows, avec XAMPP, Python, VS Code et un navigateur.
- [Annexe A11](#) : le tableau de bord de départ et son mode d'emploi. Le dossier se télécharge dans [A11_tableau_de_bord.zip](#). Il contient aussi les scripts de la base.
- [Annexe A6](#) : le jeu de données de la base.
- [Annexe A7](#) : le compte MySQL de l'application.
- [Annexe A8](#) : le fichier d'environnement, à ranger dans votre dossier personnel.

- [Annexe A9](#) : un extrait de la documentation du connecteur MySQL. Il rappelle la requête paramétrée avec `%s`.

Pour lire le service choisi dans l'adresse (`/tableau?service=atelier`), la vue utilise `request.args.get("service")`. Il renvoie `None` si l'adresse ne contient pas de service, et un texte vide `""` si on choisit « Tous les services » dans le menu. Écrivez `request.args.get("service") or None` : les deux cas donnent alors `None`.

CE QU'ON VOUS DEMANDE

Le tableau de bord a une seule page, `/tableau`. Elle affiche toutes les interventions, avec le nom de leur chantier. Un menu filtre les interventions par service.

La fonction `interventions_par_service` de la couche d'accès est à écrire. La vue qui l'appelle est à compléter. La saisie du service ne doit jamais pouvoir changer la requête envoyée à la base.

Pendant l'entretien, le formateur tapera lui-même un service piégé dans votre page. Votre tableau doit rester sain : aucune donnée qui ne devrait pas s'afficher, aucune trace d'erreur.

PAR OÙ COMMENCER

45 minutes en tout : environ 40 minutes de travail, et l'entretien de 5 minutes dès que votre tableau marche.

1. Préparez votre poste avec l'[annexe A11](#) : dossier extrait, MySQL lancé, base chargée, `technovert.env` rempli. Lancez le tableau de bord et ouvrez `/tableau`. Environ 5 min.
2. Écrivez `interventions_par_service` d'abord **sans filtre** : la requête relie l'intervention à son chantier avec `JOIN`, et renvoie tout. Environ 10 min.
3. Ajoutez le **filtre** : quand un service est donné, ajoutez `WHERE i.service = %s` et passez le service en paramètre. Environ 10 min.
4. Complétez la vue `tableau()` : elle lit le service avec `request.args.get`, appelle la couche, et calcule le total des heures. Le gabarit attend aussi `message` : c'est un texte à afficher quand le service demandé n'est pas dans `SERVICES`, sinon `None`. Environ 10 min.
5. Testez avec un service normal, puis avec une injection de votre choix. Environ 5 min.

CE QUI DOIT MARCHER À LA FIN

- ☐ Sans filtre, le tableau affiche les 12 interventions, avec le nom de chaque chantier.
 - ☐ Le filtre « atelier » n'affiche que les interventions de l'atelier.
 - ☐ Le total des heures affiché correspond aux lignes montrées.
 - ☐ Une saisie qui contient une apostrophe ne renvoie aucune ligne et ne provoque aucune trace.
 - ☐ Vous savez expliquer chaque ligne de votre requête et de votre vue.
-

Bloqué ? Dites-le au formateur pendant l'entretien. Savoir expliquer où l'on bloque fait partie du professionnalisme.

Tous les documents du module

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

↓ Télécharger A1_mini_site.zip

copie : U6-S2-M5, [annexe A1](#)

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

copie : U6-S2-M5, [annexe A2](#)

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>

Gabarit	Donnée à passer	Ce qu'elle contient
base.html	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

copie : U6-S2-M5, [annexe A3](#)

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.

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 `cloture`. 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
<code>id</code>	le numéro de l'intervention
<code>id_chantier</code>	le numéro du chantier
<code>technicien</code>	l'identifiant de la personne, au format <code>pnom</code>
<code>service</code>	<code>atelier</code> , <code>bureau etudes</code> ou <code>commercial</code>
<code>date_intervention</code>	la date, au format AAAA-MM-JJ
<code>duree_heures</code>	la durée, en heures, nombre décimal
<code>description</code>	ce qui a été fait
<code>cloturee</code>	<code>true</code> si l'intervention est terminée
<code>blocage</code>	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

copie : U6-S2-M5, [annexe A4](#)

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.

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 — L'application de départ et son mode d'emploi

↓ Télécharger [A5_technovert_web.zip](#)

copie : U6-S2-M6, [annexe A1](#)

Source : application écrite pour le module. Le mini-site vient du travail sur le développement web ; il est fourni ici en entier, prêt à brancher sur la base. Fichier [A5_technovert_web.zip](#) à télécharger ci-dessus. Utilisée dans : LABO parties A à D.

Ce que contient le dossier

technovert-web/	
app.py	les routes et les vues : vous y changez une ligne
d'import	
acces_bdd.py	la couche d'accès à la base : c'est ici que vous
écrivez	
donnees.py	l'ancienne couche, sur fichier : gardée pour comparer,
non modifiée	
chantiers.json	les anciennes données sur fichier : plus utilisées
après la bascule	
templates/	les gabarits HTML : fournis
static/style.css	la feuille de style : fournie
technovert.env.exemple	le modèle de fichier d'environnement (annexe A8)
A6_technovert.sql	le jeu de données de la base (annexe A6)
A7_compte_appli.sql	le compte MySQL de l'application (annexe A7)
A10_verifier_base.py	le script de vérification (annexe A10)

Installer le dossier sur votre poste

1. Téléchargez [A5_technovert_web.zip](#) depuis cette annexe.
2. Faites un clic droit sur le fichier, puis « Extraire tout ». Avec « Parcourir », choisissez le dossier « Documents ».
3. Ouvrez le dossier `Documents\technovert-web` dans VS Code, puis un terminal : menu « Terminal », « Nouveau terminal ».

Pour repartir de zéro, supprimez le dossier, extrayez de nouveau le zip, et rejouez

[A6_technovert.sql](#).

Lancer MySQL et l'application

1. Dans le panneau de contrôle de XAMPP, cliquez sur « Start » en face de « MySQL ». Le nom passe au vert.
2. Dans le terminal de VS Code, tapez `python -m flask --app app run --debug`. Si `python` n'est pas reconnu, tapez `py`.
3. Ouvrez `http://127.0.0.1:5000/chantiers` dans le navigateur. On arrête le serveur avec Ctrl+C.

Rendre la commande mysql disponible

La commande `mysql` est dans le dossier `C:\xampp\mysql\bin`. Pour qu'elle marche dans tous les terminaux, tapez ces deux lignes **une seule fois**, dans PowerShell, puis fermez et rouvrez VS Code :

```
$u = [Environment]::GetEnvironmentVariable("Path", "User")
[Environment]::SetEnvironmentVariable("Path", "$u;C:\xampp\mysql\bin", "User")
```

Test : `mysql --version` affiche une version de MariaDB.

Le fichier acces_bdd.py de départ

Les premières lignes chargent vos identifiants avec `load_dotenv`. Les fonctions `ouvrir()` et `lire_chantiers()` sont à écrire. La fonction `interventions_du_technicien`, tout en bas, est volontairement vulnérable : vous la corrigez en partie C.

```

"""Couche d'accès aux données de TechnoVert : tout le SQL de l'application est ici.

Les vues de app.py n'écrivent jamais de SQL. Elles appellent ces fonctions.
"""

import os
from pathlib import Path

import mysql.connector
from dotenv import load_dotenv
from werkzeug.security import check_password_hash

# Les identifiants sont dans votre dossier personnel, hors du code.
load_dotenv(Path.home() / "technovert.env")

SERVICES = ["atelier", "bureau etudes", "commercial"]

COLONNES_CHANTIER = ("SELECT ch.id, ch.id_client, cl.nom AS client, ch.nom,
ch.adresse, "
                    "ch.date_debut, ch.date_fin_prevue, ch.statut "
                    "FROM chantier ch JOIN client cl ON cl.id = ch.id_client ")

class BaseIndisponible(Exception):
    """La base ne répond pas."""

def ouvrir():
    # À ÉCRIRE (partie A)
    pass

def lire_chantiers():
    # À ÉCRIRE (partie A)
    return []

def lire_chantier(id_chantier):
    """Un chantier, ou None s'il n'existe pas."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute(COLONNES_CHANTIER + "WHERE ch.id = %s", (id_chantier,))
    ligne = curseur.fetchone()
    connexion.close()
    return ligne

def ajouter_intervention(id_chantier, technicien, service, date_intervention,
                        duree_heures, description, blocage=None):
    """Ajoute une intervention et renvoie son numéro."""
    connexion = ouvrir()
    curseur = connexion.cursor()
    curseur.execute(

```

```

        "INSERT INTO intervention (id_chantier, technicien, service,
date_intervention, "
        "duree_heures, description, cloturee, blocage) "
        "VALUES (%s, %s, %s, %s, %s, %s, FALSE, %s)",
        (id_chantier, technicien, service, date_intervention, duree_heures,
description,
        blocage or None))
    connexion.commit()
    numero = curseur.lastrowid
    connexion.close()
    return numero

def verifier_compte(login, mot_de_passe):
    """True si le login existe et si le mot de passe est le bon."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute("SELECT empreinte FROM compte WHERE login = %s", (login,))
    ligne = curseur.fetchone()
    connexion.close()
    return ligne is not None and check_password_hash(ligne["empreinte"],
mot_de_passe)

# Écrite par l'ancien prestataire. Elle marche, mais elle est dangereuse.
# Ne la corrigez pas avant l'étape 15.
def interventions_du_technicien(login, service):
    """Les interventions d'un technicien pour un service."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    requete = ("SELECT i.date_intervention, c.nom AS chantier, i.technicien,
i.service, "
               "i.duree_heures FROM intervention i JOIN chantier c ON c.id =
i.id_chantier "
               "WHERE i.technicien = '" + login + "' AND i.service = '" +
service + "' "
               "ORDER BY i.date_intervention")
    curseur.execute(requete)
    lignes = curseur.fetchall()
    connexion.close()
    return lignes

```

A6 — Le jeu de données de la base

↓ Télécharger A6_technovert.sql

copie : U6-S2-M6, [annexe A2](#)

Source : structure d'après le document de scénario, § 5, « Schéma relationnel de référence ». Les lignes sont écrites pour le module. Fichier [A6_technovert.sql](#) à télécharger ci-dessus, aussi présent dans les dossiers de départ. Utilisée dans : LABO étapes 2, 5 et 21, MISE EN SITUATION.

Ce script efface la base `technovert`, la recrée et la remplit. On le rejoue pour revenir à un état connu. Dans le dossier du fichier :

```
mysql -u root -e "source A6_technovert.sql"
```

Le mot de passe de test de tous les comptes est `Chantier2026`. La base ne range que l'empreinte du mot de passe, jamais le mot de passe lui-même.

```

-- Base TechnoVert : structure et jeu de données de test.
-- Rejouable : chaque lancement efface la base et la recrée à l'identique.
-- Lancement, dans le dossier du fichier : mysql -u root -e "source
A6_technovert.sql"

SET NAMES utf8mb4;

DROP DATABASE IF EXISTS technovert;
CREATE DATABASE technovert CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE technovert;

CREATE TABLE client (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    ville VARCHAR(60),
    telephone VARCHAR(20),
    email VARCHAR(100)
);

CREATE TABLE chantier (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_client INT NOT NULL,
    nom VARCHAR(100),
    adresse VARCHAR(150),
    date_debut DATE,
    date_fin_prevue DATE,
    statut ENUM('ouvert', 'cloture') NOT NULL DEFAULT 'ouvert',
    FOREIGN KEY (id_client) REFERENCES client(id)
);

CREATE TABLE intervention (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_chantier INT NOT NULL,
    technicien VARCHAR(30) NOT NULL,
    service ENUM('atelier', 'bureau etudes', 'commercial') NOT NULL,
    date_intervention DATE,
    duree_heures DECIMAL(4,1),
    description VARCHAR(255),
    cloturee BOOLEAN NOT NULL DEFAULT FALSE,
    blocage VARCHAR(100) NULL,
    FOREIGN KEY (id_chantier) REFERENCES chantier(id)
);

CREATE TABLE compte (
    login VARCHAR(30) PRIMARY KEY,
    empreinte VARCHAR(255) NOT NULL
);

INSERT INTO client (id, nom, ville, telephone, email) VALUES
    (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');

INSERT INTO chantier (id, id_client, nom, adresse, date_debut, date_fin_prevue,
statut) VALUES
  (1, 1, 'Arrosage du parc de la mairie', 'Parc de la mairie, Vernay',
'2026-03-02', '2026-04-30', 'cloture'),
  (2, 2, 'Arrosage des greens 1 à 9', 'Route de Montval, Montval', '2026-04-06',
'2026-06-26', 'ouvert'),
  (3, 3, 'Éclairage des allées', '12 rue des Tilleuls, Vernay', '2026-05-11',
'2026-06-12', 'ouvert'),
  (4, 4, 'Éclairage du stade', 'Stade municipal, Bellerive', '2026-06-01',
'2026-09-30', 'ouvert'),
  (5, 2, 'Éclairage du parking', 'Route de Montval, Montval', '2026-09-07',
'2026-11-27', 'ouvert'),
  (6, 1, 'Arrosage des ronds-points', 'Avenue de la Gare, Vernay', '2026-09-14',
'2026-12-18', 'ouvert');

INSERT INTO intervention (id, id_chantier, technicien, service, date_intervention,
duree_heures, description, cloturee, blocage) VALUES
  (1, 1, 'ngarcia', 'atelier', '2026-03-03', 6.0, 'Pose des programmeurs', TRUE,
NULL),
  (2, 1, 'ngarcia', 'atelier', '2026-03-10', 7.5, 'Raccordement des vannes', TRUE,
NULL),
  (3, 1, 'yropars', 'commercial', '2026-04-28', 1.0, 'Visite de réception avec le
client', TRUE, NULL),
  (4, 2, 'lmarchetti', 'bureau etudes', '2026-04-02', 3.0, 'Relevé des pentes des
greens', TRUE, NULL),
  (5, 2, 'lmoreau', 'atelier', '2026-04-07', 8.0, 'Tranchées et pose des tuyaux',
TRUE, NULL),
  (6, 2, 'lmoreau', 'atelier', '2026-05-19', 4.0, 'Réglage des arroseurs', FALSE,
'acces refuse'),
  (7, 3, 'ngarcia', 'atelier', '2026-05-12', 5.5, 'Pose des bornes lumineuses',
FALSE, NULL),
  (8, 3, 'yropars', 'commercial', '2026-05-20', 0.5, 'Point d'avancement avec le
syndic', FALSE, 'client absent'),
  (9, 4, 'lmarchetti', 'bureau etudes', '2026-06-02', 2.5, 'Calcul de l'éclairage
du terrain', TRUE, NULL),
  (10, 4, 'lmoreau', 'atelier', '2026-06-15', 8.0, 'Montage des mâts', FALSE,
NULL),
  (11, 5, 'yropars', 'commercial', '2026-09-08', 1.0, 'Signature du devis sur
place', FALSE, NULL),
  (12, 6, 'ngarcia', 'atelier', '2026-09-15', 6.0, 'Pose des goutteurs', FALSE,
NULL);

-- Mot de passe de test de tous les comptes : Chantier2026. Seule l'empreinte est
rangée.
INSERT INTO compte (login, empreinte) VALUES
  ('kbenali',

```

```
'pbkdf2:sha256:1000000$0UkaJXqPPzn8eBRD$777cc1cadfd0aaaf8922f784fb4fd7df9518e94b9b4fa894552b7bf6
('yropars',
'pbkdf2:sha256:1000000$6ZEROCdDqgIhstCD$9ea8f945a738982e98d73214d52eb78212e1247ffb3d549cd726d702e9
('lmarchetti',
'pbkdf2:sha256:1000000$DONiyJYIQZcnyZ3M$af59cc4fdb251362d0a509b3456488116c429628be176867cf7b04fa
('ngarcia',
'pbkdf2:sha256:1000000$26To0JU2Eavjsw6a$67c3369e9af86e1869331e65c6dd39d04c09dd174fd77a3b07e08fc
('lmoreau',
'pbkdf2:sha256:1000000$JhIzqxAL0VVgGcJ$92cc58a9dc617191d58a4c623d66b276cf2e67e6b395d9049ec975d1c
```

A7 — Le compte MySQL de l'application

↓ Télécharger [A7_compte_appli.sql](#)

copie : U6-S2-M6, [annexe A3](#)

Source : compte écrit pour le module. Fichier [A7_compte_appli.sql](#) à télécharger ci-dessus, aussi présent dans les dossiers de départ. Utilisée dans : LABO étapes 2 et 4, MISE EN SITUATION.

L'application ne se connecte jamais avec le compte administrateur. Elle utilise un compte à droits réduits : lire, ajouter, modifier. Il ne peut ni supprimer de table ni changer la structure.

```
mysql -u root -e "source A7_compte_appli.sql"
```

```
-- Compte MySQL de l'application TechnoVert, pour le labo seulement.
-- Lancement, dans le dossier du fichier : mysql -u root -e "source
A7_compte_appli.sql"
-- En production, le mot de passe est choisi à l'installation.
-- Il n'est écrit que dans le fichier d'environnement du serveur, jamais dans le
code.

-- Deux lignes par compte : MySQL distingue « localhost » et « 127.0.0.1 ».
CREATE USER IF NOT EXISTS 'tv_appli'@'localhost' IDENTIFIED BY 'Labo-Appli-2026';
CREATE USER IF NOT EXISTS 'tv_appli'@'127.0.0.1' IDENTIFIED BY 'Labo-Appli-2026';
GRANT SELECT, INSERT, UPDATE ON technovert.* TO 'tv_appli'@'localhost';
GRANT SELECT, INSERT, UPDATE ON technovert.* TO 'tv_appli'@'127.0.0.1';
```

A8 — Le fichier d'environnement à remplir

copie : U6-S2-M6, [annexe A4](#)

Source : modèle écrit pour le module, fourni dans les dossiers de départ sous le nom `technovert.env.exemple` . Utilisée dans : LABO étape 4, MISE EN SITUATION.

Ce fichier contient les identifiants de la base et la clé secrète de l'application. Il ne va jamais dans le dossier du code : il se range dans votre dossier personnel, `C:\Users\` suivi de votre nom. Les autres comptes de la machine ne le lisent pas.

L'application le lit seule au démarrage, avec la ligne `load_dotenv(Path.home() / "technovert.env")` de `acces_bdd.py` . Rien à charger à la main, même dans un nouveau terminal.

Dans le dossier de l'application :

```
Copy-Item technovert.env.exemple $HOME\technovert.env
notepad $HOME\technovert.env
```

Le modèle à recopier :

```
# Fichier d'environnement de l'application : modèle à recopier.
# Il se range dans votre dossier personnel, jamais dans le dossier du code.
# Copie, dans PowerShell : Copy-Item technovert.env.exemple $HOME\technovert.env
# Ouverture pour le remplir : notepad $HOME\technovert.env
TV_BDD_HOTE=127.0.0.1
TV_BDD_UTILISATEUR=tv_appli
TV_BDD_MOT_DE_PASSE=a-replacer
TV_BDD_BASE=technovert
TV_CLE_SECRETE=a-replacer-par-une-longue-valeur-au-hasard
```

- `TV_BDD_MOT_DE_PASSE` : le mot de passe du compte `tv_appli` de l'[annexe A7](#).
- `TV_CLE_SECRETE` : une longue valeur au hasard. Elle signe le cookie de session.

A9 — Extrait de la documentation du connecteur MySQL

copie : U6-S2-M6, [annexe A5](#)

Source : documentation de `mysql-connector-python`, pages « Connecting » et « Cursor.execute ». Extrait traduit et raccourci pour le module. Utilisée dans : LABO étapes 10 et 15, MISE EN SITUATION.

Ouvrir une connexion. `mysql.connector.connect()` reçoit `host` , `user` , `password` , `database` . Il lève `mysql.connector.Error` si la connexion échoue. L'option `connection_timeout` fixe le temps d'attente, en secondes.

Le curseur. `connexion.cursor(dictionary=True)` crée un curseur qui renvoie chaque ligne sous forme de dictionnaire, avec les noms de colonnes comme clés.

Exécuter une requête paramétrée. On écrit `%s` à chaque endroit où une valeur entre. On passe les valeurs à part, dans un tuple, en second argument.

```
curseur.execute("SELECT * FROM chantier WHERE statut = %s", (statut,))
```

- Le connecteur remplace chaque `%s` par la valeur, en la traitant comme une donnée, jamais comme du SQL.
- Le `%s` ne prend pas d'apostrophes autour, même pour un texte. Le connecteur les ajoute.
- Pour une seule valeur, le tuple s'écrit `(valeur,)`, avec une virgule.

Lire et écrire.

- `curseur.fetchall()` renvoie toutes les lignes d'un `SELECT`, dans une liste.
- `curseur.fetchone()` renvoie une seule ligne, ou `None`.
- Après un `INSERT`, un `UPDATE` ou un `DELETE`, il faut `connexion.commit()` pour garder le changement.
- `curseur.lastrowid` donne le numéro de la dernière ligne ajoutée.

Fermer. `connexion.close()` ferme la connexion. On la ferme dès que la requête est finie.

A10 — Le script de vérification de la base

↓ Télécharger [A10_verifier_base.py](#)

copie : U6-S2-M6, [annexe A6](#)

Source : script écrit pour le module. Fichier [A10_verifier_base.py](#) à télécharger ci-dessus, aussi présent dans le dossier de départ du labo. Utilisée dans : LABO étapes 21 et 22.

Ce script se branche sur la base par la couche d'accès. Il compte les lignes, teste les cas connus, et vérifie qu'une injection reste sans effet. Il affiche OK ou ÉCHEC sur chaque ligne.

```
python A10_verifier_base.py
```

Il lit lui aussi `technovert.env`. On le lance après avoir rejoué le jeu de données de l'[annexe A6](#).

```

"""Vérifie la base TechnoVert et la couche d'accès, avec le jeu de données de test.

À lancer dans le dossier de l'application :
    python A10_verifier_base.py

Les identifiants sont lus dans technovert.env, rangé dans votre dossier personnel.

Chaque ligne affiche OK ou ÉCHEC. Le jeu de test doit avoir été rejoué juste
avant.
"""
import sys

import acces_bdd

resultats = []

def verifier(nom, condition):
    resultats.append(condition)
    print(("OK      " if condition else "ÉCHEC  ") + nom)

def compter(table):
    connexion = acces_bdd.ouvrir()
    curseur = connexion.cursor()
    curseur.execute("SELECT COUNT(*) FROM " + table)    # nom de table fixe, jamais
saisi
    nombre = curseur.fetchone()[0]
    connexion.close()
    return nombre

try:
    acces_bdd.ouvrir().close()
except acces_bdd.BaseIndisponible as erreur:
    print("ÉCHEC  connexion à la base :", erreur)
    print("Vérifiez que MySQL est lancé dans le panneau XAMPP, et le contenu de
technovert.env.")
    sys.exit(1)
except KeyError as erreur:
    print("ÉCHEC  variable d'environnement absente :", erreur)
    print("Vérifiez que technovert.env existe dans votre dossier personnel, et
qu'il est complet.")
    sys.exit(1)

verifier("connexion à la base", True)
verifier("4 clients", compter("client") == 4)
verifier("6 chantiers", compter("chantier") == 6)
verifier("12 interventions", compter("intervention") == 12)
verifier("5 comptes", compter("compte") == 5)

chantiers = acces_bdd.lire_chantiers()

```

```

verifier("lire_chantiers() renvoie 6 chantiers", len(chantiers) == 6)
verifier("chaque chantier a le nom de son client",
        all(c.get("client") for c in chantiers))
premier = acces_bdd.lire_chantier(1)
verifier("le chantier 1 est clôturé", premier is not None and premier["statut"] ==
"cloture")
verifier("le chantier 99 n'existe pas", acces_bdd.lire_chantier(99) is None)
verifier("le compte kbenali accepte son mot de passe",
        acces_bdd.verifier_compte("kbenali", "Chantier2026"))
verifier("le compte kbenali refuse un mauvais mot de passe",
        not acces_bdd.verifier_compte("kbenali", "faux"))
verifier("ngarcia a 4 interventions pour l'atelier",
        len(acces_bdd.interventions_du_technicien("ngarcia", "atelier")) == 4)

try:
    lignes = acces_bdd.interventions_du_technicien("ngarcia", "x' OR '1'='1")
    verifier("une saisie avec des apostrophes ne renvoie aucune ligne", len(lignes)
== 0)
except Exception as erreur:
    verifier("une saisie avec des apostrophes ne provoque pas d'erreur SQL (" +
        type(erreur).__name__ + ")", False)

print()
print(f"{sum(resultats)} vérification(s) sur {len(resultats)} réussie(s).")
sys.exit(0 if all(resultats) else 1)

```

A11 — Le tableau de bord de départ et son mode d'emploi

↓ Télécharger A11_tableau_de_bord.zip

copie : U6-S2-M6, [annexe A7](#)

Source : application écrite pour le module. Fichier [A11_tableau_de_bord.zip](#) à télécharger ci-dessus. Utilisée dans : MISE EN SITUATION.

Ce que contient le dossier

tableau-de-bord/	
app.py	la vue tableau() à compléter
acces_bdd.py	la fonction interventions_par_service() à écrire
templates/	les gabarits, dont tableau.html : fournis
static/style.css	la feuille de style : fournie
technovert.env.exemple	le modèle de fichier d'environnement (annexe A8)
A6_technovert.sql	le jeu de données de la base (annexe A6)
A7_compte_appli.sql	le compte MySQL de l'application (annexe A7)

Préparer le poste

1. Extrayez `A11_tableau_de_bord.zip` dans « Documents » : « Extraire tout », puis « Parcourir ». Ouvrez le dossier `Documents\tableau-de-bord` dans VS Code, avec un terminal.
2. Lancez MySQL dans XAMPP.
3. Rechargez la base, même si elle existe déjà depuis le labo : `mysql -u root -e "source A6_technovert.sql"`, puis `mysql -u root -e "source A7_compte_appli.sql"`.
4. Si `technovert.env` n'est pas déjà dans votre dossier personnel, suivez l'[annexe A8](#).
5. Lancez le tableau de bord : `python -m flask --app app run --debug`, puis ouvrez `http://127.0.0.1:5000/tableau`.

La fonction à écrire

`interventions_par_service(service=None)` renvoie les interventions, avec le nom de leur chantier. Si on donne un service, seulement celles de ce service. Si on ne donne rien, toutes.

Chaque ligne renvoyée doit avoir ces clés, car le gabarit les lit : `date_intervention`, `chantier`, `technicien`, `service`, `duree_heures`, `blocage`. Le nom du chantier doit s'appeler `chantier` : écrivez `c.nom AS chantier` dans la requête. Une clé mal nommée ne donne aucun message d'erreur : la colonne reste vide.

La vue à compléter

`tableau()` lit le service dans l'adresse, appelle la couche, puis passe au gabarit la liste, le total des heures et le service choisi. Le gabarit `tableau.html` attend : `services`, `service`, `lignes`, `total`, `message`.