

Le Mur des Bonnes Résolutions

Objectifs

Votre mission : Concevoir une mini-application web en Python3/Flask avec une base SQLite et des pages HTML/CSS très simples. L'objectif est de construire, pas à pas, un petit mur public de résolutions 2026 : chacun peut déposer une résolution, encourager celles des autres et suivre leur progression, sans compte ni inscription.



Figure 1: L'application finale

Mise en place de l'environnement

Il vous est demandé :

- de créer un dépôt git de travail dans votre répertoire *home* : `git init ~/webbd`
- de vous placer dans ce dépôt git : `cd ~/webbd`
- de déballer le paquet dans votre dépôt : `git pull /NAS_TNCY/2526EXAMWEB/webbd.bundle main`
- de mettre en place un environnement virtuel pour votre développement (`python3 -m venv venv`)
- d'activer cet environnement (`source venv/bin/activate`)
- d'installer les dépendances indiquées dans le fichier `requirements.txt` : `pip install -r requirements.txt`.
- d'installer playwright (*optionnel*) : `playwright install`

⚠ **Attention** Vous ne devez pas ajouter et committer le contenu du répertoire contenant votre environnement virtuel.

⚠ **Important** Le code principal de votre serveur web devra se trouver dans un fichier Python dénommé `app.py`. Les *templates* doivent se trouver dans le répertoire `templates/` et les CSS (si nécessaire) dans `static/`.

⚠ **Conseil** Pensez à committer régulièrement votre progression. Cela peut nous permettre de tester une ancienne version qui fonctionnait dans le cas où votre dernier commit casserait toute votre application...

Informations sur l'évaluation

Lors de la correction de votre examen, nous devons être capables d'exécuter votre serveur en utilisant les commandes suivantes (vous n'avez pas à exécuter ces commandes !) :

```
# creation d'un environnement virtuel spécifique à l'évaluation
$ python3 -m venv venv
$ source venv/bin/activate

# installation des dépendances
$ pip install -r requirements.txt

# installation de playwright (optionnel, mais utile pour les tests)
$ playwright install

# exécution du serveur sur le port 8080 de l'hôte local
$ flask run --host=0.0.0.0 --port=5454
```

Thème et fonctionnalités

Bienvenue dans “Le Mur des Bonnes Résolutions 2026” : un tableau de voeux de début d’année où l’on écrit ses promesses, on applaudit les autres, et on voit ce qui tient... ou pas. L’application doit permettre :

- d’ajouter une résolution avec un pseudo et un texte court,
- de voter avec un bouton “🔥 je valide”,
- de changer le statut d’une résolution : en cours, tenue, abandonnée,
- d’afficher un emoji selon le statut :
 - active -> 🎯
 - kept -> ✅
 - dropped -> 😞
- d’afficher un badge “Top résolution” si le score atteint 5 votes ou plus,
- d’afficher un message d’encouragement aléatoire après un ajout,
- d’exporter toutes les résolutions en JSON via une route spécifique /api/resolutions.

Données de test (version en mémoire)

Pour la première version, utilisez une structure Python simple (liste de dictionnaires) à recopier dans le code de votre application.

```
resolutions = [  
    {  
        "id": 1,  
        "pseudo": "G.O.",  
        "texte": "Rendre ses notes rapidement",  
        "votes": 3,  
        "statut": "active",  
        "created_at": "2025-01-05",  
    },  
    {  
        "id": 2,  
        "pseudo": "Pierre L.",  
        "texte": "Faire des examens plus faciles",  
        "votes": 6,  
        "statut": "dropped",  
        "created_at": "2025-01-02",  
    },  
    {  
        "id": 3,  
        "pseudo": "Mr. Bou",  
        "texte": "Rire en silence",  
        "votes": 1,  
        "statut": "kept",  
        "created_at": "2025-01-03",  
    },  
]
```

Partie 1 - Une application sans base de données

👉 **Astuce** À tout moment, vous pouvez exécuter le script `./check_expected_files.sh` afin de vérifier que vous avez bien réalisé les fichiers attendus.

⚙️ **Tests** Si vous avez réussi à installer playwright, vous pouvez exécuter des tests de votre application en utilisant la commande `pytest`.

Question 1 - Route de vérification

Créer une route GET `/status` qui retourne uniquement la chaîne :

Le mur est prêt !

Question 2 - Première page HTML (données en mémoire)

Créer une page HTML servie sur la route `/` pour une requête GET qui affiche la liste des résolutions (données en dur). Cette page doit être rendue avec un template (`templates/index.html`).

Préciser le titre de la page (Le mur des bonnes résolutions) et afficher ce titre comme un titre de niveau 1.

Afficher dans une liste à puces (liste non ordonnée) les résolutions.

Pour chaque résolution, afficher :

- Le statut avec l'emoji correspondant (dans une balise `span` de classe `resolution-emoji`),
- le pseudo de l'auteur de résolution (dans une balise `strong` de classe `pseudo`),
- le texte de la résolution (dans une balise `span` de classe `resolution-content`),
- le nombre de votes pour cette résolution (dans une balise `span` de classe `vote`),
- sa date de saisie (dans une balise `span` de classe `date`),
- un badge "Top résolution" si le nombre de votes est supérieur ou égale à 5 (dans une balise `span` de classe `badge`).

⚠️ **Remarque** Pour le moment, vous pouvez ignorer les 3 boutons permettant de voter ou changer le statut d'une résolution.

En guise de barre de navigation, ajouter deux liens hypertextes en haut de la page ("Accueil" pointant sur l'URI `/` et "Ajouter" pointant sur l'URI `/add`)

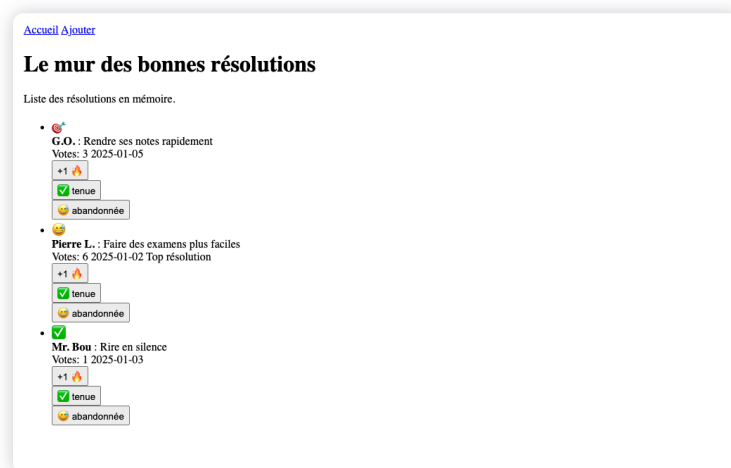


Figure 2: Capture Question 2 - /

Question 3 - Page d'ajout

Créer une page répondant à une requête GET sur la route `/add` avec un formulaire pour ajouter une résolution. Cette page doit être rendue avec un template (`templates/add.html`).

La page doit comporter :

- Un titre de niveau 1 dont l'intitulé est "Ajouter une résolution"
- un formulaire avec deux champs texte :
 - l'un dénommé `pseudo` (texte court),
 - l'autre dénommé `texte` (résolution).

Lors d'une requête POST sur la route /add, vous traiterez le formulaire soumis pour ajouter la résolution en mémoire avec un nouvel identifiant unique (le prochain entier disponible). Pour cette nouvelle résolution, son statut sera active, son nombre de votes nul, et la date de création (created_at) sera la date courante sous forme d'une chaîne de caractères au format YYYY-MM-DD (que vous pouvez obtenir par un appel à `date.today().isoformat()`).

Après ajout, afficher à nouveau le formulaire suivi d'un message (dans un paragraphe HTML dont la classe sera message). Ce message sera sélectionné de manière aléatoire parmi une petite liste prédéfinie (["Courage, on y croit !", "Bonne chance !", "Go !"]). Vous pouvez utiliser la fonction Python `random.choice(some_list)` qui permet de tirer aléatoirement une valeur parmi une liste de valeurs passée en paramètre (n'oubliez pas le `import random`).

Figure 3: Capture Question 3 - /add

Question 4 - Voter et changer le statut (en mémoire)

Ajouter des boutons/formulaires sur la page d'accueil pour respectivement :

- voter pour une résolution particulière d'identifiant `id` en effectuant une requête POST sur la route `/vote/<int:id>`,
- marquer une résolution (d'identifiant `id`) comme "tenue" (kept) en effectuant une requête POST sur la route `/status/<int:id>/kept`,
- marquer une résolution (d'identifiant `id`) comme "abandonnée" (dropped) en effectuant une requête POST sur la route `/status/<int:id>/dropped`.

Après l'action prise en compte, rediriger l'utilisateur vers la route `/`.

Figure 4: Capture Question 4 - actions

Question 5 - Export JSON

Ajouter une route `/api/resolutions` à votre application. Une requête GET sur cette route retournera la liste complète des résolutions au format JSON.

```
[
  {
    "created_at": "2025-01-05",
    "id": 1,
    "pseudo": "G.O.",
    "statut": "active",
    "texte": "Rendre ses notes rapidement",
    "votes": 3
  },
  ...
]
```

```
{
  "created_at": "2025-01-02",
  "id": 2,
  "pseudo": "Pierre L.",
  "statut": "dropped",
  "texte": "Faire des examens plus faciles",
  "votes": 6
},
{
  "created_at": "2025-01-03",
  "id": 3,
  "pseudo": "Mr. Bou",
  "statut": "kept",
  "texte": "Rire en silence",
  "votes": 1
},
{
  "created_at": "2025-12-26",
  "id": 4,
  "pseudo": "Zoe",
  "statut": "active",
  "texte": "Lire 10 pages",
  "votes": 0
}
]
```

Partie 2 - Stockage des données dans une base relationnelle (SQLite)

Dans cette partie, vous allez modifier votre application pour utiliser une base SQLite, **sans casser la version précédente**. La version “mémoire” doit rester testable, et la version SQLite doit être accessible via **de nouvelles routes**.

⚠ **Conseil** Lorsque vous arrivez ici, c’est une bonne idée de committer la version fonctionnelle de votre application.

Question 6 - Modèle relationnel (SQLite)

Proposer un schéma relationnel simple permettant de stocker les informations concernant les résolutions (identifiant, auteur, texte, nombre de votes, statut, date de création).

Votre schéma doit garantir l’unicité d’une résolution, que les nombre des votes soient toujours positifs et que le statut soit contraint aux 3 valeurs possibles (active, kept ou dropped).

Sauvegarder le schéma dans un fichier `database_schema.txt` sous la forme de requête(s) de création de la (ou les) table(s).

Créer le fichier `resolutions.db` à la racine du dépôt et insérer les données de test (cette partie peut être réalisée à la main ou en Python selon votre convenance).

Question 7 - Page principale SQLite

Reproduire la page d’accueil listant toutes les résolutions issues de la base de données SQLite que l’on peut accéder par une requête GET sur la route `/db`. Vous utiliserez un template dédié (`/templates/db_index.html`).

Vous changerez l’intitulé de la page (titre de niveau 1) pour y indiquer : Le mur des bonnes résolutions (SQLite).

Question 8 - Ajout (version SQLite)

Reproduire la page et les fonctionnalités d’ajout d’une résolution (cette fois-ci dans la base de données) via une route dédiée (`/db/add`) et un template dédié (`/templates/db_add.html`).

Vous changerez l’intitulé de la page (titre de niveau 1) pour y indiquer (SQLite) comme sur la figure.



Figure 5: Capture Question 7 - `/db/add`

Question 9 - Vote (version SQLite)

Ajouter la fonctionnalité de vote (via SQLite) sur une route dédiée (`/db/vote/<int:id>`) pour une requête POST.

Question 10 - Changement de statut (version SQLite)

Ajouter les actions de changement de statut (via SQLite) sur des routes dédiées (`/db/status/<int:id>/kept` et `/db/status/<int:id>/dropped`).

Question 11 - Export JSON (version SQLite)

Ajouter la fonctionnalité d’exportation (via SQLite) au format JSON sur une route dédiée (GET `/db/api/resolutions`)

Question 12 - Top des résolutions

Ajouter une page obtenue en retour à une requête GET sur la route `/db/top` qui affiche uniquement les résolutions avec un nombre de votes supérieur ou égal à 5. Cette page sera rendu en utilisant un template spécifique `templates/db_top.html`.

Préciser le titre de la page (Top résolutions (SQLite)) et afficher ce titre comme un titre de niveau 1.

Afficher dans une liste à puces (liste non ordonnée) les résolutions classées par ordre décroissant de nombre de votes.

Pour chaque résolution, afficher :

- Le statut avec l’emoji correspondant (dans une balise `span` de classe `resolution-emoji`),
- le pseudo de l’auteur de résolution (dans une balise `strong` de classe `pseudo`),

- le texte de la résolution (dans une balise `span` de classe `resolution-content`),
- le nombre de votes pour cette résolution (dans une balise `span` de classe `vote`),

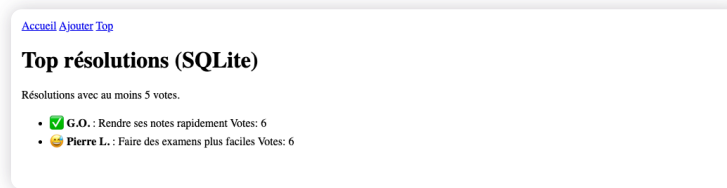


Figure 6: Capture Question 12 - /db/top

Question 13 - Un peu de coloriage

Votre application n'est pas très jolie pour le moment. Faites-vous plaisir et ajouter du style ;)

Vous trouverez ci-dessous un exemple d'ajout de styles (comme toujours, les goûts et les couleurs ne se discutent pas).



Figure 7: L'application finale

Vérification finale et rendu (5 min avant la fin de l'épreuve)

Si vous êtes arrivé jusqu'ici, votre dépôt doit contenir a minima les fichiers suivants :

- ☒ requirements.txt
- ☒ app.py
- ☒ resolutions.db
- ☒ database_schema.txt
- ☒ les fichiers de vos *templates* et de vos ressources statiques (CSS par exemple).

Exécutez le script `./check_expected_files.sh`

Pensez à ajouter et committer tous les fichiers nécessaires au fonctionnement de votre application (code principal, templates, fichiers statiques, base de données, etc.)

Exécutez le script `./build_submission.sh`

Recopier le fichier `rendu-<VOTRE_LOGIN>.bundle` dans le répertoire sur le NAS `/NAS_TNCY/exams/etudiants/exam_<VOTRE_LOGIN_UL>/`

Pour plus de sûreté, recopier également tout votre répertoire de travail `~/webbd` dans le répertoire sur le NAS `/NAS_TNCY/exams/etudiants/exam_<VOTRE_LOGIN_UL>/`. La commande devrait ressembler à :

```
cp -r ~/webbd /NAS_TNCY/exams/etudiants/exam_<VOTRE_LOGIN_UL>/
```

Bravo !

