Faculté des Sciences
Aix-Marseille Université

Année universitaire 2023/2024

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☐ 1er semestre ☒ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L2 Nom du diplôme : Licence d'Informatique
Code du module : SIN4U03TL Libellé du module : WEB
Calculatrices autorisées : NON Documents autorisés : OUI

Les calculatrices, téléphones, ordinateurs, etc. sont interdits. Les documents sont autorisés.

Site de questions et réponses

On veut implémenter une application web permettant à ses utilisateurs de poser des questions et d'apporter des réponses aux questions posées par d'autres utilisateurs. Tous les utilisateurs peuvent consulter la liste des questions posées et, pour chaque question, la liste des réponses apportées. Les utilisateurs identifiés (c'est à dire possédant un compte sur l'application et connectés à ce compte) peuvent en outre poser de nouvelles questions et proposer de nouvelles réponses aux questions déjà posées. De plus, les utilisateurs identifiés peuvent voter pour (+1) ou contre (-1) chaque question et chaque réponse. Pour chaque question et pour chaque réponse, un utilisateur émet au plus un vote (pour ou contre). Les questions et réponses sont affichées sur l'application (pour tous les utilisateurs, identifiés ou non) par ordre décroissant de popularité (les plus populaires d'abord). La popularité d'une question ou d'une réponse est calculée en sommant les votes associés à cette question ou réponse.

On implémentera cette application en utilisant le framework flask et une base de données sqlite3 pour la partie serveur. On considère l'architecture modèle/vues/contrôleur suivante pour cette application.

1. *Modèle de données.* L'application utilise quatre tables SQL :
   (a) `user` avec comme colonnes une clé primaire auto-incrémentée `id` et des colonnes de texte `name` et `password_hash`
   (b) `question` avec comme colonnes une clé primaire auto-incrémentée `id`, des colonnes de texte `question` et `time_stamp` et une colonne `user_id` contenant des entiers
   (c) `answer` avec comme colonnes une clé primaire auto-incrémentée `id`, des colonnes de texte `answer` et `time_stamp` et des colonnes `user_id` et `question_id` contenant des entiers
   (d) `question_votes` avec comme colonnes une clé primaire auto-incrémentée `id` et des colonnes `user_id`, `question_id` et `vote` contenant des entiers
   (e) `answer_votes` avec comme colonnes une clé primaire auto-incrémentée `id` et des colonnes `user_id`, `answer_id` et `vote` contenant des entiers

   Les fonctions d'accès fournies par le modèle sont :
   (a) `get_all_questions()` qui renvoie une liste contenant un dictionnaire pour chaque question dans la table `question`, chaque dictionnaire contenant les paires clés/valeurs suivantes : `id`/identifiant de la question, `question`/texte de la question, `user_id`/identifiant utilisateur associé, `user_name`/nom utilisateur associé, `time_stamp`/date associée, `nb_votes`/somme des votes associés (notez que `user_name` et `nb_votes` ne sont pas stockés directement dans la table `question`)
   (b) `get_question(id)` qui renvoie un couple `question, answers`, où `question` est un dictionnaire pour la question d'identifiant `id` contenant les même clés/valeurs que pour la fonction précédente et `answer` est une liste contenant un dictionnaire pour chaque réponse à la question d'identifiant `id` présente dans la table `answer`. Chaque dictionnaire dans la liste `answers` contient les paires clés/valeurs suivantes : `id`/identifiant de la réponse, `answer`/texte de la réponse, `user_id`/identifiant utilisateur associé, `user_name`/nom utilisateur associé, `time_stamp`/date associée, `nb_votes`/somme des votes associés
   (c) `register_question(question, user_id)` qui enregistre la question dont le texte est `question` posée par l'utilisateur d'identifiant `user_id` dans la table `question` en lui associant la date actuelle (année, jour, mois, heure, etc.) relativement au fuseau horaire UTC
   (d) `register_answer(question_id, answer, user_id)` comme la fonction précédente, mais pour enregistrer une réponse à la question d'identifiant `question_id` dans la table `answer`.

(e) `register_vote_on_question(question_id, user_id, vote)` qui : s'il existe une entrée dans la table `question_votes` pour la question d'identifiant `question_id` et l'utilisateur d'identifiant `user_id`, remplace la valeur dans la colonne `vote` pour cette entrée par l'entier `vote` passé en argument ; sinon crée une entrée dans la table `question_votes` pour la question d'identifiant `question_id` et l'utilisateur d'identifiant `user_id` avec la valeur `vote` dans la colonne `vote`

(f) `register_vote_on_answer(answer_id, user_id, vote)` qui réalise la même opération que la fonction précédente mais pour compter les votes pour les réponses, dans la table `answer_votes`.

(g) `login(user, password)` qui retourne -1 si aucun utilisateur de nom `user` n'est présent dans la colonne `name` de la table `user` ou si le mot de passe fourni n'est pas compatible avec le hash de mot de passe associé au nom `user` dans la table `user` ; qui retourne l'identifiant (colonne `id`) associé à l'utilisateur `user` dans la table `user` sinon

(h) `new_user(user, password)` qui retourne -1 si un utilisateur de nom `user` existe déjà dans la table `user` et qui sinon crée un nouvel utilisateur dans cette table avec le nom et mot de passe fournis et renvoie l'identifiant correspondant

2. *Vues.* Quatre vues, réalisées par le biais de templates jinja2.

(a) Une page d'accueil `index.html` affichant :
- pour les utilisateurs identifiés : un message de bienvenue contenant leur nom d'utilisateur ; un lien permettant de se déconnecter ; une partie avec un titre invitant à poser une nouvelle question et un formulaire utilisant la méthode `POST` pour permettre à l'utilisateur d'entrer sa question dans un champ de texte
- pour les utilisateurs non-identifiés : des liens vers les routes `/login` et `/new_user`
- pour tous les utilisateurs : une partie avec un titre approprié, contenant la liste de toutes les questions précédemment posées par ordre décroissant de nombre de votes, avec pour chaque question la somme des votes associés à cette question et un lien affichant le texte de la question et renvoyant vers la page dédiée à cette question

(b) Une page d'affichage d'une question `question.html` avec :
- pour les utilisateurs identifiés : un message de bienvenue contenant leur nom d'utilisateur et un lien de déconnexion
- pour les utilisateurs non-identifiés des liens permettant de se connecter et de créer un nouvel utilisateur
- pour tous les utilisateurs : un lien vers la page d'accueil ; un titre ; le texte de la question avec la somme des votes associés, le nom de l'utilisateur ayant posé la question et la date d'enregistrement de la question dans la base de données ; une partie contenant les réponses avec un titre approprié et la liste des réponses, triées par ordre décroissant de nombre de votes, avec, pour chaque réponse, le texte de la réponse, la somme des votes associés, le nom de l'utilisateur ayant soumis la réponse et la date d'enregistrement de la réponse

De plus, pour les utilisateurs identifiés, la page affichera un formulaire utilisant la méthode `POST` permettant de fournir une nouvelle réponse, ainsi que, pour la question et pour chaque réponse, deux formulaires utilisant la méthode `POST` : un pour voter pour et un pour voter contre

(c) Une page `login.html` contenant un formulaire utilisant la méthode `POST` et permettant à un utilisateur existant de se connecter

(d) Une page `new_user.html` contenant un formulaire utilisant la méthode `POST` et permettant à un nouvel utilisateur de se créer un compte

3. *Routes et logique de contrôle*

(a) La route `GET /` affiche la page d'index.

(b) La route `GET /questions/<id>` affiche la page de la question d'identifiant `id`.

(c) La route `POST /question` n'est accessible qu'aux utilisateurs identifiés et traite les données du formulaire de la vue `index.html` permettant de poser une nouvelle question. Elle redirige vers la page associée à la question nouvellement enregistrée.

(d) La route `POST /answer` n'est accessible qu'aux utilisateurs identifiés et traite les données du formulaire de la vue `question.html` permettant de proposer une nouvelle réponse

(e) La route `POST /vote` n'est accessible qu'aux utilisateurs identifiés et traite les données des formulaires de la vue `question.html` permettant aux utilisateurs enregistrés de voter (que ce soit un vote pour ou un vote contre et que ce soit pour une question ou pour une réponse).

(f) les routes `POST /login`, `GET /new_user`, `GET /logout`, permettent respectivement aux utilisateurs d'accéder aux formulaires de connection et de création d'utilisateur et de se déconnecter

(g) les routes `POST /login` et `POST /new_user` permettent de traiter les données des formulaires correspondants

Questions

Le barème est donné à titre indicatif et est susceptible de changer lors de la correction si nécessaire.

1. (1 point) On considère le fichier `create_db.py` contenant le code suivant :

```python
import sqlite3
DBFILENAME = 'db.sqlite'
def db_run(query, args=(), db_name=DBFILENAME):
  with sqlite3.connect(db_name) as conn:
    cur = conn.execute(query, args)
    conn.commit()
```

   Écrire le code à ajouter au fichier `create_dbs.py` pour initialiser la table `question`.

2. (1 point) On considére le fichier `data_model.py` contenant le code suivant :

```python
import sqlite3
from werkzeug.security import generate_password_hash, check_password_hash
import datetime
DBFILENAME = 'db.sqlite'

# Utility functions
def db_fetch(query, args=(), all=False, db_name=DBFILENAME):
  with sqlite3.connect(db_name) as conn:
    # to allow access to columns by name in res
    conn.row_factory = sqlite3.Row
    cur = conn.execute(query, args)
    # convert to a python dictionary for convenience
    if all:
      res = cur.fetchall()
      if res:
        res = [dict(e) for e in res]
      else:
        res = []
    else:
      res = cur.fetchone()
      if res:
        res = dict(res)
  return res

def db_insert(query, args=(), db_name=DBFILENAME):
  with sqlite3.connect(db_name) as conn:
    cur = conn.execute(query, args)
    conn.commit()
    return cur.lastrowid

def db_run(query, args=(), db_name=DBFILENAME):
  with sqlite3.connect(db_name) as conn:
    cur = conn.execute(query, args)
    conn.commit()

def db_update(query, args=(), db_name=DBFILENAME):
  with sqlite3.connect(db_name) as conn:
    cur = conn.execute(query, args)
    conn.commit()
    return cur.rowcount

def register_question(question, user_id):
  time = datetime.datetime.now(datetime.UTC)
  time = f"{time} (UTC)"
```

   Compléter le code de la fonction `register_question` pour qu'elle ait le comportement attendu.

3. (3 points) Écrire le code à ajouter au fichier `data_model.py` pour implémenter la fonction `get_all_questions`. Conseil : écrivez des fonctions séparées `get_nb_question_votes(id)` et `get_username(user_id)` que vous utiliserez dans votre fonction principale pour obtenir respectivement la somme des votes associés à la question d'identifiant `id` et le nom d'utilisateur associé à l'utilisateur d'identifiant `user_id`.

4. (2 points) On considère l'implémentation ci-dessous pour la fonction `new_user`

```
def new_user(user, password):
  result = db_fetch('SELECT id FROM user WHERE name = ?',
                    (user,))
  if result is None:
    result = db_insert('INSERT INTO user (name, password_hash) VALUES (?, ?)',
                       (user, password))
    return result
  else:
    return -1
```

où `user` et `password` sont respectivement le nom d'utilisateur et le mot de passe fournis par l'utilisateur dans le formulaire de création d'un nouvel utilisateur. Cette implémentation pose-t-elle des problèmes de sécurité ? Si oui, décrivez explicitement un scénario d'attaque permis par cette implémentation et décrivez comment modifier le code pour corriger la vulnérabilité.

5. (2 points) Dans la suite on considère que toutes les fonctions du modèle de données ont été implémentée et qu'elles peuvent donc être utilisées librement. On considère le fichier `server.py` contenant le code suivant :

```
from flask import Flask, session, Response, request, redirect, url_for, render_template
import data_model as model
from functools import wraps
app = Flask(__name__)
app.secret_key = b'6dbb6b3863634aa6a72270de16df48e666f2564fddcc5fe3c27effe4393a7f4b'

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not('user_id' in session):
            return Response("Veuillez vous connecter pour accéder à cette ressource",
                            status=401)
        else:
            return f(*args, **kwargs)
    return decorated_function
```

Écrire le code à ajouter au fichier `server.py` pour implémenter les routes `GET /login` et `POST /login`. On supose que les noms associés au nom d'utilisateur et au mot de passe entrés par l'utilisateur par le formulaire de la vue `login.html` sont respectivement `user` et `password`.

6. (4 points) Proposer un template jinja2 pour la vue `index.html` et écrire le code à ajouter au fichier `server.py` pour implémenter la routes `GET /`. On pourra utiliser la commande :
`l = sorted(l, key=lambda d: d[key], reverse=True)` pour obtenir une version de la liste de dictionnaires `l` triée par ordre décroissant des valeurs des dictionnaires associées à la clé `key`.

7. (2 points) Écrire le code à ajouter à `server.py` pour implémenter la route `POST /question`.

8. (4 points) Proposez un fichier javascript `sort.js` à distribuer avec l'application et permettant à l'utilisateur de cliquer sur un bouton sur la page d'accueil pour changer l'ordre dans lequel les questions sont triées : si elles étaient triées par nombre de votes, elle deviennent triées par ordre de nouveauté (la plus récente d'abord) et vice-versa. Vous pouvez supposer que vous disposez d'une fonction javascript `to_seconds` transformant la chaîne de caractère obtenue par le biais du code python

```
time = datetime.datetime.now(datetime.UTC)
time = f"{time} (UTC)"
```

en un nombre de secondes écoulées depuis une date fixe. Il est possible que vous deviez modifier le template `index.html`, dans ce cas indiquez brièvement modifications nécessaires.

9. (1 point) Écrivez un fichier CSS pour cette application web et expliquer comment s'assurer que ce fichier est bien utilisé sur la vue `index.html` et est bien distribué aux clients de l'application.