

老師，因為我是用公司帳號使用 GPT，沒有辦法分享連結，只能以大致截圖的方式讓您看，非常抱歉。

Prompt 1:

我想要用 python flask 的套件，讓使用者對某個問題進行投票，並顯示實時的投票結果。

架構：設計投票主頁面，包含投票的項目、內容和點擊投票按鈕。

| | | | |
|--------|----|--------|--------|
| 投票項目 | 內容 | 目前投票結果 | 點擊進入投票 |
| 投票項目 1 | 內容 | | |
| 投票項目 2 | 內容 | | |

以上為大致頁面格式，你可以修改，大致長這樣就好。

首頁顯示所有投票問題的列表，並在右下角新增「新增投票問題」按鈕，允許使用者自行設置投票問題和選項，並將資料儲存至資料庫。

Prompt 2:

加入 SQLAlchemy 和 RESTful API，實現數據儲存與操作：使用 SQLAlchemy 來將新增的投票與投票結果儲存到資料庫，並建立 RESTful API 以提供數據操作功能。

架構：

點擊新增投票問題，進入新增投票頁面，可自由新增投票項目與投票選項

投票選項可以是多選也可以是單選，依新增者自行設定，儲存後將資料儲存到資料庫裡，並且回到所有投票問題頁面。

使用 Flask 和 SQLAlchemy 建立資料庫，並將投票數據持久化存儲。

該投票的創立者可自由刪除投票問題和結果，實現數據的完全控制。

Prompt 3:

擴展前端功能，提供投票和詳細頁面 當點擊投票按鈕時進入投票頁面，使用者可以進行投票，投票後刷新頁面 更新當前投票結果。

Prompt 4:

加入使用者登入功能，實現使用者註冊和登入功能

架構：

設置註冊和登入頁面，使用者需登入後才能操作投票問題。

使用 SQLAlchemy 建立使用者表格，包含使用者資訊。

Prompt 5:

讓投票問題刪除紐只有當時的創建者可以做刪除，非該創建者的頁面不會出現刪除紐。

Prompt 1:



這是一個簡單的投票應用程式，用 Flask 和 SQLite 資料庫來實作。使用者可以查看投票列表、進行投票，並即時看到投票結果。此外，還提供一個「新增投票問題」的功能，讓使用者可以創建新的投票問題和選項。

以下是這個應用程式的基本架構：

- 1. 首頁：顯示所有投票問題，包含各投票問題的選項、內容和當前的投票結果。
- 2. 投票頁面：使用者可以點選進行投票並即時更新結果。
- 3. 新增投票頁面：允許使用者新增投票問題及選項。

Flask 應用程式程式碼

- 1. 首先，創建資料庫模型（SQLite）來儲存投票問題和選項。
- 2. 然後，建立 Flask 應用程式並設置路由來處理顯示、投票和新增投票的邏輯。

建立資料庫模型

python 複製程式碼

```
# models.py
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Poll(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.String(255), nullable=False)
    options = db.relationship('Option', backref='poll', lazy=True)

class Option(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    option_text = db.Column(db.String(100), nullable=False)
    votes = db.Column(db.Integer, default=0)
    poll_id = db.Column(db.Integer, db.ForeignKey('poll.id'), nullable=False)
```

主應用程式程式碼

python

複製程式碼

```
# app.py
from flask import Flask, render_template, redirect, url_for, request
from models import db, Poll, Option

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)

@app.route('/')
def index():
    polls = Poll.query.all()
    return render_template('index.html', polls=polls)

@app.route('/vote/<int:poll_id>', methods=['GET', 'POST'])
def vote(poll_id):
    poll = Poll.query.get_or_404(poll_id)
    if request.method == 'POST':
        option_id = request.form['option']
```



複製程式碼

```
@app.route('/vote/<int:poll_id>', methods=['GET', 'POST'])
def vote(poll_id):
    poll = Poll.query.get_or_404(poll_id)
    if request.method == 'POST':
        option_id = request.form['option']
        option = Option.query.get(option_id)
        option.votes += 1
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('vote.html', poll=poll)

@app.route('/add_poll', methods=['GET', 'POST'])
def add_poll():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        options = request.form.getlist('options')
        new_poll = Poll(title=title, content=content)
        db.session.add(new_poll)
        db.session.flush() # 獲取 new_poll 的 id
        for option_text in options:
            option = Option(option_text=option_text, poll_id=new_poll.id)
            db.session.add(option)
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('add_poll.html')

if __name__ == '__main__':
    app.run(debug=True)
```

HTML 頁面設計

1. 首頁 (index.html)：顯示所有投票問題及投票結果。
2. 投票頁面 (vote.html)：選擇並投票特定選項。
3. 新增投票頁面 (add_poll.html)：設置新的投票問題和選項。

以下是範例的 HTML 模板程式碼：

index.html

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>投票首頁</title>
</head>
<body>
  <h1>所有投票問題</h1>
  <table>
    <tr>
      <th>投票項目</th>
      <th>內容</th>
      <th>目前投票結果</th>
      <th>進入投票</th>
    </tr>
    {% for poll in polls %}
    <tr>
      <td>{{ poll.title }}</td>
      <td>{{ poll.content }}</td>
      <td>
        {% for option in poll.options %}
        {{ option.option_text }}: {{ option.votes }} 票<br>
        {% endfor %}
      </td>
      <td><a href="{{ url_for('vote', poll_id=poll.id) }}">投票</a></td>
    </tr>
    {% endfor %}
  </table>
  <a href="{{ url_for('add_poll') }}">新增投票問題</a>
</body>
</html>
```

vote.html

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{ poll.title }}</title>
</head>
<body>
  <h1>{{ poll.title }}</h1>
  <p>{{ poll.content }}</p>
  <form method="POST">
    {% for option in poll.options %}
    <input type="radio" name="option" value="{{ option.id }}"> {{ option.option_text }}
    {% endfor %}
    <button type="submit">投票</button>
  </form>
</body>
</html>
```

add_poll.html

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>新增投票</title>
</head>
<body>
  <h1>新增投票問題</h1>
  <form method="POST">
    <label>標題：<input type="text" name="title"></label><br>
    <label>內容：<textarea name="content"></textarea></label><br>
    <label>選項：</label><br>
    <input type="text" name="options"><br>
    <input type="text" name="options"><br>
    <input type="text" name="options"><br>
    <button type="submit">新增投票</button>
  </form>
</body>
</html>
```

說明

1. 首頁顯示所有投票問題，並提供各投票的當前結果及進入投票頁面的連結。
2. 投票頁面讓使用者選擇一個選項並提交。
3. 新增投票頁面允許使用者設置投票標題、內容和選項。

這個應用程式提供了基礎的投票功能，包含顯示投票、進行投票和即時顯示結果等功能，可以根據需求進行擴展。

```
from flask import Flask, render_template, redirect, url_for, request
from models import db, Poll, Option
```

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)
```

```
@app.route('/')
def index():
    polls = Poll.query.all()
    return render_template('index.html', polls=polls)
```

```
@app.route('/vote/<int:poll_id>', methods=['GET', 'POST'])
def vote(poll_id):
    poll = Poll.query.get_or_404(poll_id)
    if request.method == 'POST':
        option_id = request.form['option']
        option = Option.query.get(option_id)
        option.votes += 1
        db.session.commit()
        return redirect(url_for('index'))
    return render_template('vote.html', poll=poll)
```

```
@app.route('/add_poll', methods=['GET', 'POST'])
def add_poll():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        options = request.form.getlist('options')
        new_poll = Poll(title=title, content=content)
        db.session.add(new_poll)
        db.session.flush() # 獲取 new_poll 的 id
        for option_text in options:
            option = Option(option_text=option_text,
                             poll_id=new_poll.id)
            db.session.add(option)
            db.session.commit()
        return redirect(url_for('index'))
    return render_template('add_poll.html')
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

app.py的程式碼



看來你的 `app.py` 程式碼已經設置好 Flask 應用程式了，但你需要確保資料庫的資料表已經正確創建。以下是一些步驟，確保你的應用程式能夠正常啟動且不會出現資料表缺失的錯誤。

1. 檢查模型文件 (`models.py`)

確認你的 `models.py` 文件中包含以下代碼，用於定義 `Poll` 和 `Option` 資料表：

```
python 複製程式碼

from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Poll(db.Model):
    __tablename__ = 'poll'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.String(255), nullable=False)
    options = db.relationship('Option', backref='poll', lazy=True)

class Option(db.Model):
    __tablename__ = 'option'
    id = db.Column(db.Integer, primary_key=True)
    option_text = db.Column(db.String(100), nullable=False)
    votes = db.Column(db.Integer, default=0)
    poll_id = db.Column(db.Integer, db.ForeignKey('poll.id'), nullable=False)
```

2. 初始化資料庫

在首次執行時，你需要初始化資料庫並創建所有資料表。可以在 Python shell 中執行以下指令，或者在 `app.py` 中暫時加上一段代碼來執行。

在 Python shell 中執行（確保已啟動虛擬環境並在項目根目錄）：

```
python 複製程式碼

from app import db
db.create_all()
```

或者，在 `app.py` 的末尾加上以下代碼，然後執行一次應用程式（之後可以刪除這段代碼）：

或者，在 `app.py` 的末尾加上以下代碼，然後執行一次應用程式（之後可以刪除這段代碼）：

```
python

with app.app_context():
    db.create_all()
```

這樣可以確保 `Poll` 和 `Option` 表格已經正確創建。

3. 再次啟動應用程式

初始化完資料庫後，請重新啟動 Flask 應用程式，並檢查是否能正常顯示首頁及進行投票。

總結

這樣做應能解決 `no such table: poll` 的錯誤。再確認一次主要的步驟：

1. 確保 `models.py` 中的表格模型正確無誤。
2. 執行 `db.create_all()` 來創建資料表。

如果還有任何問題，請隨時告訴我。



還是一樣



如果還是出現同樣的錯誤，可能是資料表沒有成功創建或資料庫連接有問題。請按照以下步驟進一步排查：

1. 確認 `db.create_all()` 是否執行成功

嘗試直接在 `app.py` 中執行 `db.create_all()` 並確認是否有錯誤訊息。例如：

```
python

from flask import Flask, render_template, redirect, url_for, request
from models import db, Poll, Option

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)

# 新增這段來確保資料庫表格創建
with app.app_context():
    db.create_all()

@app.route('/')
def index():
    polls = Poll.query.all()
    return render_template('index.html', polls=polls)

# 其餘程式碼保持不變
```

執行 `app.py` 看看是否還有錯誤。如果沒有錯誤且資料表成功創建，你應該能找到 `polls.db` 資料庫文件並看到 `poll` 和 `option` 資料表。

2. 檢查資料庫文件

確認 `polls.db` 資料庫文件是否已創建在你的項目根目錄中。如果該文件不存在，表示資料庫未正確初始化。可以手動刪除該文件後重新執行 `db.create_all()`。

3. 刪除並重新創建資料庫

有時候資料庫文件可能已損壞或被部分創建，可以嘗試刪除 `polls.db` 然後重新創建。確保已關閉所有連接到資料庫的程式，然後刪除 `polls.db`，重新執行應用程式以重新創建資料表。

4. 驗證資料表是否存在（可選）

如果你熟悉 SQLite，可以使用 SQLite CLI 或 SQLite Browser 等工具打開 `polls.db`，檢查是否有 `poll` 和 `option` 表格。如果這些表格已存在，可能是程式碼中的查詢出現錯誤；如果表格不存在，則需重新執行 `db.create_all()`。

Prompt 2:

| | |
|--|--|
| <p>為了實現你的需求，我們需要以下幾個步驟：</p> <ol style="list-style-type: none">1. 更新資料庫模型：為了支持單選和多選選項，我們可以在 <code>Poll</code> 模型中新增一個 <code>is_multiple_choice</code> 欄位。2. 新增 RESTful API：使用 Flask 的藍圖和 Flask-RESTful 來提供 RESTful API，支持新增、查詢、更新和刪除投票項目。3. 更新頁面和功能：新增前端頁面來支持投票創建和刪除功能。 <p>以下是具體的實作步驟：</p> <h3>1. 更新資料庫模型</h3> <p>在 <code>models.py</code> 中更新 <code>Poll</code> 模型，並且設置多選/單選的控制。</p> | <pre>from flask_sqlalchemy import SQLAlchemy db = SQLAlchemy() class Poll(db.Model): __tablename__ = 'poll' id = db.Column(db.Integer, primary_key=True) title = db.Column(db.String(100), nullable=False) content = db.Column(db.String(255), nullable=False) is_multiple_choice = db.Column(db.Boolean, default=False) # 單選或多選控制 options = db.relationship('Option', backref='poll', lazy=True, cascade="all, delete-orphan") class Option(db.Model): __tablename__ = 'option' id = db.Column(db.Integer, primary_key=True) option_text = db.Column(db.String(100), nullable=False) votes = db.Column(db.Integer, default=0) poll_id = db.Column(db.Integer, db.ForeignKey('poll.id'), nullable=False)</pre> |
| <h3>2. 建立 RESTful API</h3> <p>在 <code>api.py</code> 中定義 API 端點來實現新增、查詢、更新和刪除投票的功能。</p> | |
| <pre>from flask import Blueprint, jsonify, request from models import db, Poll, Option api_bp = Blueprint('api', __name__) @api_bp.route('/polls', methods=['POST']) def create_poll(): data = request.json title = data.get('title') content = data.get('content') is_multiple_choice = data.get('is_multiple_choice', False) options = data.get('options', []) new_poll = Poll(title=title, content=content, is_multiple_choice=is_multiple_choice) db.session.add(new_poll) db.session.flush() # 獲取 new_poll 的 id for option_text in options: option = Option(option_text=option_text, poll_id=new_poll.id) db.session.add(option) db.session.commit() return jsonify({'message': 'Poll created successfully'}), 201</pre> | <pre>@api_bp.route('/polls/<int:poll_id>', methods=['DELETE']) def delete_poll(poll_id): poll = Poll.query.get_or_404(poll_id) db.session.delete(poll) db.session.commit() return jsonify({'message': 'Poll deleted successfully'}), 200 @api_bp.route('/polls', methods=['GET']) def get_polls(): polls = Poll.query.all() polls_data = [{ 'id': poll.id, 'title': poll.title, 'content': poll.content, 'is_multiple_choice': poll.is_multiple_choice, 'options': [{ 'id': option.id, 'text': option.option_text, 'votes': option.votes } for option in poll.options] } for poll in polls] return jsonify(polls_data), 200</pre> |

3. 更新主應用程式

在 app.py 中引入 api_bp 並註冊藍圖。這樣可以讓 RESTful API 在主應用程式中可用。

```
from flask import Flask, render_template, redirect, url_for, request
from models import db, Poll, Option
from api import api_bp

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)

app.register_blueprint(api_bp, url_prefix='/api') # 註冊 API 藍圖

with app.app_context():
    db.create_all()

@app.route("/")
def index():
    polls = Poll.query.all()
    return render_template('index.html', polls=polls)

# 其餘程式碼保持不變
```

4. 更新前端模板

新增投票頁面 (add_poll.html) 支持設置單選或多選選項。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <title>新增投票</title>
</head>
<body>

  <h1>新增投票問題</h1>

  <form method="POST" action="/api/polls" id="pollForm">

    <label>標題：<input type="text" name="title"></label><br>

    <label>內容：<textarea name="content"></textarea></label><br>

    <label>是否多選：<input type="checkbox" name="is_multiple_choice" value="true"></label><br>

    <label>選項：</label><br>

    <input type="text" name="options"><br>
    <input type="text" name="options"><br>
    <input type="text" name="options"><br>

    <button type="submit">新增投票</button>

  </form>
  <script>
    document.getElementById("pollForm").onsubmit = async function(event) {
      event.preventDefault();
      const formData = new FormData(event.target);
      const data = {
        title: formData.get("title"),
        content: formData.get("content"),
        is_multiple_choice: formData.get("is_multiple_choice") ? true : false,
        options: formData.getAll("options").filter(o => o)
      };
      const response = await fetch('/api/polls', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
```



```
});

if (response.ok) {

    window.location.href = "/";

}

};

</script>

</body>

</html>
```

投票問題頁面 (index.html)

提供刪除按鈕來讓創建者刪除投票。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

    <title>投票首頁</title>

</head>
<body>

    <h1>所有投票問題</h1>

    <table>

        <tr>

            <th>投票項目</th>

            <th>內容</th>

            <th>目前投票結果</th>

            <th>操作</th>

        </tr>

        {% for poll in polls %}

        <tr>

            <td>{{ poll.title }}</td>

            <td>{{ poll.content }}</td>

            <td>

                {% for option in poll.options %}

                {{ option.option_text }}: {{ option.votes }} 票<br>

                {% endfor %}

            </td>

        </tr>
```

```
        </td>

        <td>

            <a href="/vote/{{ poll.id }}">投票</a>

            <button onclick="deletePoll('{{ poll.id }}')">刪除</button>

        </td>

    </tr>

    {% endfor %}

</table>

<a href="{{ url_for('add_poll') }}">新增投票問題</a>

<script>
    async function deletePoll(pollId) {

        const response = await fetch(`/api/polls/${pollId}`, { method: 'DELETE' });

        if (response.ok) {

            location.reload();

        }

    }

</script>
```

5. 測試應用程式

現在，你可以測試以下功能：

- 新增投票問題，設置單選或多選。
- 透過 API 儲存投票數據並顯示在首頁。
- 創建者可以刪除投票問題。

這些功能應該可以實現你的需求，讓使用者能夠自由新增、投票和刪除投票問題。

Debug

進入http://127.0.0.1:5000/add_poll沒有東西

如果在進入 `http://127.0.0.1:5000/add_poll` 時沒有顯示任何內容，可能的原因有：

- `add_poll.html` 模板文件不存在：請檢查 `templates` 資料夾中是否有 `add_poll.html` 文件。
- 路由函數返回值有問題：確保 Flask 的路由函數正確地返回了模板。

檢查和更新 `add_poll` 路由

在 `app.py` 中，確保 `add_poll` 路由函數如以下代碼所示：

```
@app.route('/add_poll', methods=['GET', 'POST'])
def add_poll():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        options = request.form.getlist('options')
        is_multiple_choice = request.form.get('is_multiple_choice', 'off') == 'on'

        # 創建新的投票

        new_poll = Poll(title=title, content=content, is_multiple_choice=is_multiple_choice)
        db.session.add(new_poll)

        db.session.flush() # 獲取 new_poll 的 id

        for option_text in options: # 添加選項

            option = Option(option_text=option_text, poll_id=new_poll.id)
            db.session.add(option)

        db.session.commit()

        return redirect(url_for('index'))

    # 確保返回 `add_poll.html`

    return render_template('add_poll.html')
```

檢查 add_poll.html 模板

在 templates 資料夾中創建 add_poll.html 文件，並確保它包含一個表單來填寫投票標題、內容、選項，以及選擇單選或多選的設置。以下是一個範例模板：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <title>新增投票</title>
</head>
<body>

  <h1>新增投票問題</h1>

  <form method="POST" action="/add_poll">

    <label>標題：<input type="text" name="title" required></label><br>

    <label>內容：<textarea name="content" required></textarea></label><br>

    <label>是否多選：<input type="checkbox" name="is_multiple_choice"></label><br>

    <label>選項：</label><br>

    <input type="text" name="options" required><br>
    <input type="text" name="options"><br>
    <input type="text" name="options"><br>

    <button type="submit">新增投票</button>

  </form>
</body>
</html>
```

注意事項

- 1. 確認路徑：檢查你的 add_poll.html 文件是否位於 templates 資料夾下。
- 2. 重新啟動 Flask 應用程式：在進行模板和路由修改後，請重新啟動 Flask 應用程式，以確保所有更改已加載。
- 3. 檢查終端的錯誤訊息：如果還是無法顯示，檢查終端的錯誤日誌，看看是否有提示找不到模板或路徑錯誤的問題。

網頁可以執行了 但點擊新增投票按鈕沒有反應

如果點擊「新增投票」按鈕沒有反應，可能的原因包括：

- 1. 表單未正確提交：按鈕未觸發表單提交。
- 2. 後端未收到請求：可能是路徑或表單資料的格式有問題。
- 3. JavaScript 錯誤 (如果使用 JavaScript 處理提交) 。

排查步驟

1. 確認表單的提交按鈕

檢查 `add_poll.html` 表單內的按鈕是否為 `type="submit"`，這樣它會觸發表單提交。請確認按鈕代碼如下：

html

`<button type="submit">新增投票</button>`

複製程式碼

這樣確保按鈕會提交整個表單。

2. 檢查表單的 `action` 屬性

確認表單的 `action` 指向正確的路徑。如果 `add_poll` 路由的 URL 是 `/add_poll`，那麼表單應該如下所示：

html

`<form method="POST" action="/add_poll">
 <!-- 表單內容 -->
</form>`

複製程式碼

↓

4. 使用開發者工具檢查請求和錯誤訊息

在瀏覽器中打開開發者工具（通常按 `F12`），切換到「Network」選項卡並點擊「新增投票」按鈕，檢查是否有 `POST` 請求發送到 `/add_poll`。若無請求，可能是按鈕或表單結構有誤。如果有錯誤，檢查「Console」選項卡的錯誤訊息。

5. 檢查後端日誌

在終端中查看 Flask 的日誌，確認 Flask 是否收到表單的 `POST` 請求，以及是否有錯誤訊息。

最終範例程式碼

`add_poll.html`

3. 檢查後端 `POST` 請求的處理

在 `app.py` 的 `/add_poll` 路由中，請確認 `POST` 方法已正確處理並重定向到首頁。

python

```
@app.route('/add_poll', methods=['GET', 'POST'])
def add_poll():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        options = request.form.getlist('options')
        is_multiple_choice = request.form.get('is_multiple_choice', 'off') == 'on'

        # 創建新的投票
        new_poll = Poll(title=title, content=content, is_multiple_choice=is_multiple_choice)
        db.session.add(new_poll)
        db.session.flush() # 獲取 new_poll 的 id

        # 添加選項
        for option_text in options:
            option = Option(option_text=option_text, poll_id=new_poll.id)
            db.session.add(option)

        db.session.commit()
        return redirect(url_for('index'))

    return render_template('add_poll.html')
```

複製程式碼

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <title>新增投票</title>

</head>
<body>

  <h1>新增投票問題</h1>

  <form method="POST" action="/add_poll">

    <label>標題：<input type="text" name="title" required></label><br>
```

```
<label>內容：<textarea name="content" required></textarea></label><br>

<label>是否多選：<input type="checkbox" name="is_multiple_choice"></label><br>

<label>選項：</label><br>

<input type="text" name="options" required><br>
<input type="text" name="options"><br>
<input type="text" name="options"><br>

<button type="submit">新增投票</button>

</form>
</body>
</html>
```

```
"C:\Users\zhen9\AppData\Local\Programs\Python\Python311\Lib
\site-packages\sqlalchemy\engine\default.py", line 941, in
do_execute
    cursor.execute(statement, parameters)
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) no
such column: poll.is_multiple_choice
[SQL: SELECT poll.id AS poll_id, poll.title AS poll_title, poll.content
AS poll_content, poll.is_multiple_choice AS poll_is_multiple_choice
FROM poll]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

這個錯誤 `sqlite3.OperationalError: no such column: poll.is_multiple_choice` 表示 `is_multiple_choice` 欄位尚未成功添加到資料庫中的 `poll` 表格。這可能是因為資料庫未更新，或者資料庫遷移未成功執行。

以下是解決方案：

1. 刪除舊的資料庫並重新創建

如果你的應用程式還在開發階段，而且沒有重要的資料，你可以刪除現有的 `polls.db` 資料庫並重新創建。這樣會避免手動遷移的麻煩。

步驟如下：

- 刪除 `polls.db` 文件。
- 確保你的模型定義是最新的，包含 `is_multiple_choice` 欄位。
- 重新啟動 Flask 應用程式，並使用 `db.create_all()` 來重新創建資料庫。

在 `app.py` 的代碼末尾添加這段代碼來重新創建資料庫：

```
python

with app.app_context():
    db.create_all()
```

執行完一次之後，你可以移除這段代碼。

2. 使用 Flask-Migrate 進行資料庫遷移（如果你需要保留現有資料）

如果你不想刪除現有資料庫，可以使用 Flask-Migrate 來進行遷移。以下是步驟：

安裝 Flask-Migrate

如果尚未安裝，請執行：

```
bash

pip install Flask-Migrate
```

設置 Flask-Migrate

在 `app.py` 中設置 Flask-Migrate：

```
python

from flask_migrate import Migrate
migrate = Migrate(app, db)
```


進行資料庫遷移

在終端執行以下命令來遷移資料庫：

```
# 初始化遷移

flask db init

# 創建遷移腳本

flask db migrate -m "Add is_multiple_choice column to Poll model"

# 執行遷移

flask db upgrade
```

這樣就可以將 `is_multiple_choice` 欄位添加到 `poll` 表格，而不需要刪除整個資料庫。

3. 檢查資料庫

遷移完成後，重新啟動 Flask 應用程式，並再次檢查是否能夠正常執行。如果成功，則錯誤應該解決。如果還有問題，請確認 `Poll` 模型中 `is_multiple_choice` 欄位的定義，以及終端的日誌輸出。

Prompt 3:

接著擴展前端功能，提供投票和詳細頁面
當點擊投票按鈕時進入投票頁面，使用者可以進行投票

要實現這個功能，我們可以擴展前端，新增兩個新頁面：

- 1. 投票頁面：當使用者點擊「投票」按鈕時，進入一個新的頁面，顯示該投票的詳細資訊及選項，讓使用者可以選擇一個選項進行投票。
- 2. 詳細頁面：投票完成後，可以顯示投票結果。

以下是具體的實作步驟：

1. 更新 `app.py` 路由

在 `app.py` 中新增 `vote` 和 `result` 路由：

```
from flask import Flask, render_template, redirect, url_for, request
from models import db, Poll, Option

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)

# 投票頁面

@app.route('/vote/<int:poll_id>', methods=['GET', 'POST'])
def vote(poll_id):
    poll = Poll.query.get_or_404(poll_id)
    if request.method == 'POST':
        option_id = request.form['option']
        option = Option.query.get(option_id)
        option.votes += 1
        db.session.commit()
        return redirect(url_for('result', poll_id=poll_id))
    return render_template('vote.html', poll=poll)

# 詳細頁面 (投票結果)

@app.route('/result/<int:poll_id>')
def result(poll_id):
    poll = Poll.query.get_or_404(poll_id)
    return render_template('result.html', poll=poll)
```

2. 建立 vote.html (投票頁面模板)

這個頁面將顯示投票問題和選項，讓使用者選擇並進行投票。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <title>投票</title>

  <style>
    .container {
      width: 60%;
      margin: 50px auto;
      text-align: center;
      padding: 20px;
      border: 1px solid #ddd;
      border-radius: 8px;
      background-color: #f9f9f9;
    }
    h1 {
      margin-bottom: 20px;
    }
    .options {
      margin-top: 20px;
      text-align: left;
    }
    .options label {
      display: block;
      margin-bottom: 10px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>{{ poll.title }}</h1>
    <p>{{ poll.content }}</p>
    <form method="POST">
      <div class="options">
        {% for option in poll.options %}
          <label>
            <input type="radio" name="option" value="{{ option.id }}" required> {{ option.option_text }}
          </label>
        {% endfor %}
      </div>

      <button type="submit" class="submit-btn">提交投票</button>
    </form>
  </div>
</body>
</html>
```

```
.submit-btn {
  margin-top: 20px;
  padding: 10px 20px;
  font-size: 1em;

  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.submit-btn:hover {
  background-color: #0056b3;
}
</style>
</head>
<body>
  <div class="container">
    <h1>{{ poll.title }}</h1>
    <p>{{ poll.content }}</p>
    <form method="POST">
      <div class="options">
        {% for option in poll.options %}
          <label>
            <input type="radio" name="option" value="{{ option.id }}" required> {{ option.option_text }}
          </label>
        {% endfor %}
      </div>

      <button type="submit" class="submit-btn">提交投票</button>
    </form>
  </div>
</body>
</html>
```

3. 建立 result.html (投票結果頁面模板)

這個頁面會在投票完成後顯示該投票的結果。

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
.back-btn {
  margin-top: 20px;
  padding: 10px 20px;
  font-size: 1em;

  background-color: #28a745;
  color: white;
  border: none;
```

```
<meta charset="UTF-8">

<title>投票結果</title>

<style>
  .container {
    width: 60%;
    margin: 50px auto;
    text-align: center;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 8px;
    background-color: #f9f9f9;
  }
  h1 {
    margin-bottom: 20px;
  }
  .results {
    margin-top: 20px;
    text-align: left;
  }
}
```

```
    border-radius: 4px;
    cursor: pointer;
    text-decoration: none;
  }
  .back-btn:hover {
    background-color: #218838;
  }
</style>
</head>
<body>
  <div class="container">

    <h1>{{ poll.title }} - 投票結果</h1>

    <div class="results">
      {% for option in poll.options %}

        <p>{{ option.option_text }}: {{ option.votes }} 票</p>

      {% endfor %}
    </div>

    <a href="{{ url_for('index') }}" class="back-btn">返回首頁</a>

  </div>
</body>
</html>
```

4. 修改 index.html (首頁模板)

確保「投票」按鈕會導向新的 vote 路由。

在 index.html 中，將「投票」按鈕的連結更新為 /vote/{{ poll.id }}：

html

<td>
 投票
</td>

複製程式碼

5. CSS 調整 (可選)

可以根據需要調整 vote.html 和 result.html 的 CSS，以保持整體頁面風格一致。

最後

這樣的設計應該可以讓使用者點擊「投票」按鈕後，跳轉到投票頁面進行選擇，並在提交後進入結果頁面顯示投票結果。這種流程提供了完整的投票功能，試試看效果如何，若有其他需求，隨時告訴我！

| Debug | |
|---|---|
| <div>測試後我發現即便在創立投票時使用著勾選多選選項，投票時依然為單選</div> <p>要解決這個問題，可以根據投票問題的設定來決定投票時是否允許多選。如果創立投票時勾選了「多選」選項，則在投票頁面顯示為多選選項；如果未勾選，則顯示為單選。</p> <h3>1. 更新 <code>vote.html</code> (投票頁面模板)</h3> <p>在 <code>vote.html</code> 中，檢查 <code>poll.is_multiple_choice</code> 的值來決定輸入類型。若 <code>is_multiple_choice</code> 為 <code>True</code>，則顯示為多選（<code>checkbox</code>）；否則顯示為單選（<code>radio</code>）。</p> <p>更新後的 <code>vote.html</code>：</p> | <pre><div class="options"> {% for option in poll.options %} <label> {% if poll.is_multiple_choice %} <!-- 多選選項 --> <input type="checkbox" name="options" value="{{ option.id }}"> {% else %} <!-- 單選選項 --> <input type="radio" name="option" value="{{ option.id }}" required> {% endif %} {{ option.option_text }} </label> {% endfor %} </div></pre> |
| <h3>2. 更新 <code>vote</code> 路由邏輯 (在 <code>app.py</code> 中)</h3> <p>在 <code>app.py</code> 的 <code>vote</code> 路由中，根據 <code>is_multiple_choice</code> 判斷如何處理投票結果。如果是多選，需要將所有被選中的選項累加計算。</p> <pre>@app.route('/vote/<int:poll_id>', methods=['GET', 'POST']) def vote(poll_id): poll = Poll.query.get_or_404(poll_id) if request.method == 'POST': if poll.is_multiple_choice: # 多選情況，取出所有選中的選項 option_ids = request.form.getlist('options') for option_id in option_ids: option = Option.query.get(option_id) if option: option.votes += 1 else: # 單選情況，取出選中的選項 option_id = request.form['option'] option = Option.query.get(option_id) if option: option.votes += 1 db.session.commit() return redirect(url_for('result', poll_id=poll_id)) return render_template('vote.html', poll=poll)</pre> | <p>else:</p> <pre># 單選情況，取出選中的選項 option_id = request.form['option'] option = Option.query.get(option_id) if option: option.votes += 1 db.session.commit() return redirect(url_for('result', poll_id=poll_id)) return render_template('vote.html', poll=poll)</pre> <h3>說明</h3> <ol style="list-style-type: none"><code>vote.html</code> 模板：根據 <code>poll.is_multiple_choice</code> 的值來決定使用 <code>checkbox</code>（多選）還是 <code>radio</code>（單選）。<code>app.py</code> 的 <code>vote</code> 路由：<ul style="list-style-type: none">若投票為多選，則使用 <code>request.form.getlist('options')</code> 來取得所有被選中的選項，並依次累加每個選項的票數。若為單選，則使用 <code>request.form['option']</code> 來取得被選中的單個選項，並累加該選項的票數。 <p>這樣的設計應該可以解決多選和單選的需求。試試看，看看是否符合預期的投票行為！</p> |

Prompt 4:

為了實現使用者註冊和登入功能，可以按照以下步驟來進行：

1. 設置使用者模型和資料庫

首先，我們需要使用 SQLAlchemy 設定一個 `User` 模型來儲存使用者資訊，包括使用者名稱和加密的密碼。

2. 設置註冊和登入路由

設置 `register` 和 `login` 路由，並進行表單處理，包含加密密碼的存儲和驗證。

3. 設置登入保護機制

使用 Flask-Login 來管理使用者的登入狀態，確保只有登入的使用者才能進入其他頁面。

4. 設計註冊和登入頁面

使用 HTML 設置註冊和登入表單，讓使用者可以進行註冊和登入。

5. 檢查使用者的登入狀態

在所有需要登入的頁面前檢查使用者的登入狀態，若未登入則重定向至登入頁面。

以下是具體的實作步驟：

1. 更新 `models.py`，新增 `User` 模型

python

複製程式碼

```
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()

class User(db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)
```


2. 設置 Flask-Login

安裝 Flask-Login 並在 `app.py` 中進行設置。

```
bash

pip install flask-login
```

複製程式碼

在 `app.py` 中新增相關設置：

```
python

from flask import Flask, render_template, redirect, url_for, request, flash
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required,
from models import db, User

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///polls.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app)

# 設置 Flask-Login
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login' # 若未登入則重定向到 login 頁面

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

複製程式碼

3. 註冊和登入邏輯

在 `app.py` 中新增註冊和登入路由。

```
python

from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin

class User(db.Model, UserMixin): # 繼承 UserMixin
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

# 註冊路由
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # 檢查是否已存在相同的使用者名稱
        if User.query.filter_by(username=username).first():
            flash('使用者名稱已存在')
            return redirect(url_for('register'))

        new_user = User(username=username)
        new_user.set_password(password)
        db.session.add(new_user)
        db.session.commit()
        flash('註冊成功，請登入')
        return redirect(url_for('login'))
    return render_template('register.html')
```

登入路由

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            login_user(user)
            flash('登入成功')
            return redirect(url_for('index'))
        else:
            flash('無效的使用者名稱或密碼')
            return redirect(url_for('login'))
    return render_template('login.html')
```

登出路由

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('您已登出')
    return redirect(url_for('login'))
```

4. 設置保護路由

在所有需要登入的頁面前加上 `@login_required`，確保使用者必須登入才能訪問這些頁面。

python

複製程式碼

```
@app.route('/')
@login_required
def index():
    # 這裡是首頁代碼，只有登入的使用者可以訪問
```

5. 註冊和登入頁面模板

註冊頁面 (register.html)

html

複製程式碼

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>註冊</title>
</head>
<body>
    <h2>註冊</h2>
    <form method="POST">
        <label>使用者名稱:<input type="text" name="username" required></label><br>
        <label>密碼:<input type="password" name="password" required></label><br>
        <button type="submit">註冊</button>
    </form>
    <p>已經有帳號了? <a href="{{ url_for('login') }}">登入</a></p>
</body>
</html>
```

登入頁面 (login.html)

html

複製程式碼

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登入</title>
</head>
<body>
    <h2>登入</h2>
    <form method="POST">
        <label>使用者名稱:<input type="text" name="username" required></label><br>
        <label>密碼:<input type="password" name="password" required></label><br>
        <button type="submit">登入</button>
    </form>
    <p>還沒有帳號? <a href="{{ url_for('register') }}">註冊</a></p>
</body>
</html>
```

要實現「僅投票問題的創建者可刪除問題」，可以根據以下步驟進行：

1. 在 `Poll` 表中新增 `creator_id` 欄位來記錄創建者的使用者 ID。
2. 當使用者創建投票問題時，將當前登入的使用者 ID 記錄在 `creator_id` 中。
3. 在顯示頁面時，檢查當前使用者是否為創建者，如果是則顯示刪除按鈕，否則不顯示。

1. 修改 `models.py`，為 `Poll` 表新增 `creator_id`

在 `Poll` 表中新增 `creator_id` 欄位，用來儲存創建者的使用者 ID。

python

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()

class Poll(db.Model):
    __tablename__ = 'poll'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.String(255), nullable=False)
    is_multiple_choice = db.Column(db.Boolean, default=False) # 單選或多選控制
    options = db.relationship('Option', backref='poll', lazy=True, cascade="all, delete-orphan")
    creator_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) # 創建者
```

複製程式碼

2. 修改 `add_poll` 路由，在創建投票問題時記錄創建者

在 `add_poll` 路由中，將當前登入使用者的 ID 記錄為投票問題的 `creator_id`。

python

```
from flask_login import current_user

@app.route('/add_poll', methods=['GET', 'POST'])
@login_required
def add_poll():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        options = request.form.getlist('options')
        is_multiple_choice = request.form.get('is_multiple_choice', 'off') == 'on'

        # 創建新的投票，並將當前使用者 ID 記錄為創建者
        new_poll = Poll(title=title, content=content, is_multiple_choice=is_multiple_choice)
        db.session.add(new_poll)
        db.session.flush() # 獲取 new_poll 的 id
```

複製程式碼

3. 修改 `index.html`，僅在創建者登入時顯示刪除按鈕

在模板中檢查當前使用者的 ID 是否與 `creator_id` 一致，若一致則顯示刪除按鈕。

<td>

```
<!-- 僅顯示給創建者 -->
{% if poll.creator_id == current_user.id %}
    <button onclick="deletePoll('{{ poll.id }}')">刪除</button>
{% endif %}
```

</td>