**Teen Poll MVP**

**Goals**

1. Landing page: show 12 category buttons.
2. Category page: show that category’s blocks.
3. Block page: show that block’s questions.
4. Under each question: options (A, B, C, ..., OTHER).
5. On click:
   * Single-choice: record vote, render bar chart.
   * Checkbox (multi-select): record all selected with **weights = 1/(number selected)**, render weighted bar chart.
   * OTHER: show a small textarea; submit to free-text table.
6. After a selection, show a **validation message** box with a **“More?”** button that reveals **companion advice**.
7. Use PostgreSQL, React, .jsx
8. Please use the modern service: I click “Run” to run the program and you can see the logs so I don’t need to copy paste the logs
9. (Wait until all the other parts finished) When a user first comes to the landing page, after clicking on a category button, a dropdown list of years shows up with: Before 2007, 2007, 2008, …2012, after 2012. Only 2007 - 2012 selection can go to the Category page. Collect user\_uuid and year\_of\_birth.

**Data is source-of-truth**

* Use the provided schema\_setup.sql and schema\_results.sql for DDL.
* Load setup data from 4 CSVs:
  + categories.csv
  + blocks.csv
  + questions.csv
  + options.csv

**Setup CSV columns (exact)**

**Results schema (in schema\_results.sql)**

Please check the INDEX part because I am not sure about that part.

Use exactly these two tables (plus helpful indexes). Include weights for checkbox rows.

-- Single-choice  
CREATE TABLE IF NOT EXISTS responses (

CREATE INDEX IF NOT EXISTS idx\_responses\_q ON responses(question\_code);  
CREATE INDEX IF NOT EXISTS idx\_responses\_qo ON responses(question\_code, option\_code);

-- Multi-select (checkbox). Each selected option is a row with a weight.  
CREATE TABLE IF NOT EXISTS checkbox\_responses (

CREATE INDEX IF NOT EXISTS idx\_cxr\_q ON checkbox\_responses(question\_code);  
CREATE INDEX IF NOT EXISTS idx\_cxr\_qo ON checkbox\_responses(question\_code, option\_code);

-- Free text for OTHER  
CREATE TABLE IF NOT EXISTS other\_responses (

CREATE INDEX IF NOT EXISTS idx\_other\_q ON other\_responses(question\_code);

Notes:

* weight is stored at insert time for checkbox submissions: weight = 1.0 / array\_length(option\_codes, 1) (app side computes).
* For single-choice, each row represents weight 1 by definition.

BOM and multiple line problems

Please review everything below carefully before implement – I don’t understand this part.

**Backend**

**Stack:** FastAPI, SQLAlchemy Core, psycopg2 (sync is fine).

**Policy:** DDL only from the SQL files. No Python-side table creation. Use reflection or raw SQL.

**Endpoints (API contracts — copy exactly)**

* GET /api/categories
  + 200:  
    [{ "category\_id":2, "category\_name": "Friends", "sort\_order": 1 }]
* GET /api/categories/{category\_id}/blocks
  + 200:  
    [{ "block\_code": "FRIENDS\_A", "block\_name": "Making friends", "sort\_order": 1 }]
* GET /api/blocks/{block\_code}/questions
  + 200:  
    [{ "question\_code": "FRIENDS\_001", "question\_text": "How do you...", "check\_box": false, "sort\_order": 1 }]
* GET /api/questions/{question\_code}/options
  + 200:  
    [  
     {  
     "option\_select": "A",  
     "option\_text": "I walk up and say hi",  
     "validate\_text": "That counts as courage. Small steps matter.",  
     "companion\_text": "If you want more ideas: choose a time/place with low pressure, ...",  
     "sort\_order": 1  
     }  
    ]
* POST /api/vote (single-choice)
  + Body:  
    { "question\_code": "FRIENDS\_001", "option\_select": "A", "uuid": "optional-uuid" }
  + 200: { "ok": true }
* POST /api/checkbox\_vote (multi-select with weights)
  + Body:  
    { "question\_code": "FRIENDS\_MULTI\_001", "option\_selects": ["A","C"], "uuid": "optional-uuid" }
  + Behavior: let n = len(option\_selects). Insert one row per option with weight = 1.0/n.
  + 200: { "ok": true }
* POST /api/other (free text for OTHER)
  + Body:  
    { "question\_code": "FRIENDS\_001", "question\_text": "prompt shown", "other\_text": "my answer", "uuid": "optional-uuid" }
  + 200: { "ok": true }
* GET /api/results/{question\_code} (auto-detect single vs checkbox)
  + If check\_box = false:  
    { "question\_code": "FRIENDS\_001",  
     "results": [ { "option\_select": "A", "count": 12 }, { "option\_select": "B", "count": 5 } ] }
  + If check\_box = true (weighted):  
    { "question\_code": "FRIENDS\_MULTI\_001",  
     "results": [ { "option\_select": "A", "count": 7.5 }, { "option\_select": "C", "count": 3.5 } ] }

**SQL for results**

* Single-choice:

SELECT option\_select, COUNT(\*)::numeric AS count

FROM responses

WHERE question\_code = :q

GROUP BY option\_select

ORDER BY count DESC;

* Checkbox (weighted):

SELECT option\_select, COALESCE(SUM(weight), 0)::numeric AS count

FROM checkbox\_responses

WHERE question\_code = :q

GROUP BY option\_select

ORDER BY count DESC;

**Insert logic for checkbox**

App computes the weight and inserts one row per option:

# pseudo

n = len(option\_selects)

w = 1.0 / n if n > 0 else 0

for code in option\_selects:

INSERT INTO checkbox\_responses(question\_code, option\_select, weight, uuid)

VALUES (:q, :code, :w, :uuid)

**Frontend**

**Stack:** React + Vite + TypeScript. Recharts for charts. React Router for pages.

**Routing**

* / Landing: fetch /api/categories. Render 12 category buttons.
* /category/:category\_id: fetch /api/categories/{category\_id}/blocks. Render block buttons.
* /block/:block\_code: fetch /api/blocks/{block\_code}/questions. For each question:
  + fetch /api/questions/{question\_code}/options.
  + If check\_box === false:
    - Render radio buttons. On click: POST /api/vote, then GET /api/results/{question\_code}, render bar chart.
  + If check\_box === true:
    - Render checkboxes and a Submit button. On submit: POST /api/checkbox\_vote, then GET /api/results/{question\_code}, render weighted bar chart.
  + If user chooses the option with option\_label === "OTHER":
    - Show a textarea and a Submit button.
    - On submit: POST /api/other. (OTHER does not change the bar chart unless you choose to count it; for MVP, do not include it in the chart.)

**Validation message with “More?”**

* After a successful vote (single or checkbox), show a message box under the question:
  + Content = the validate\_text from the **selected option** (single-choice) or, for checkbox, show a stacked list of the selected options’ validate\_text in selection order.
  + Include a small “More?” button in the box. When clicked:
    - Reveal companion\_text (companion advice) for that option. For checkbox, reveal a collapsible section for each selected option.
* Data source: both validate\_text and companion\_text come from GET /api/questions/{question\_code}/options.

**Project layout**

backend/

main.py

config.py

db.py

import\_data.py

schema\_setup.sql

schema\_results.sql

requirements.txt

frontend/

index.html

src/

main.tsx

App.tsx

pages/

Landing.tsx

Category.tsx

Block.tsx

components/

Question.tsx

OptionsList.tsx

ValidationBox.tsx

ResultsBarChart.tsx

**Guardrails**

* DDL only from SQL files. No Python create\_all.
* Always use question\_code across API, DB, CSVs, and UI.
* For checkbox submissions, enforce weights = 1/number\_selected.
* Keep endpoints and response shapes exactly as defined above.

**Run**

Backend:

cd backend

python -m venv .venv && . .venv/bin/activate # Windows: .venv\Scripts\activate

pip install -r requirements.txt

export DATABASE\_URL=postgresql://user:pass@localhost:5432/teenpoll

export DEBUG=true

python import\_data.py

uvicorn main:app --reload --port 8000

Frontend:

cd frontend

npm create vite@latest . -- --template react-ts

npm i axios recharts react-router-dom

npm run dev # <http://localhost:5173>

That’s it — this gives Eric everything needed to scaffold cleanly, with checkbox weighting and the validation/companion flow built-in from day one.