

CQOx_gen 拡張案③



これだと何もなっていないですね。



Thought for 1m 49s >

そう、その感覚は正しいです。

今の画面は「きれいなデモ」にはなっているけれど、世界を何ひとつ更新していない状態です。

ここから「ちゃんと意味のあるツール」にするには、

スライダーを動かすたびに

- ① ユーザーの世界状態がどこか1カ所でも変わる
- ② その結果が次回以降の提案や推定に反映される

この2つを満たす必要があります。

1. 何が「なっていない」のかを整理する

スクショを見る限り、今の実装はだいたいこうなっているはずです：

- Layer A: 今の状態スライダー
- Layer B: 何を大事にしたいかスライダー
- Layer C: 準備プランのシミュレーション（スライダー+予測バー）
- 下部に「このデモは emotion_cqox_sample_5000.csv を元にはしています」という説明

現状の問題点：

1. 世界状態が変わっていない
 - スライダーを動かしても、DB に Episode も Preference も保存されていない。
 - 毎回「初期状態からやり直し」で、ユーザーの人生の履歴が一切積み上がらない。
2. 予測が「誰のものでもない数字」になっている
 - サンプルCSVの係数 or ハードコードされた式に従うだけなら、「一般論として準備すれば良さそうですね」で終わる。
 - 「あなたが過去にこういう準備をしたとき、こうなってきた」という文脈がゼロ。
3. 辛い人への負荷が減っていない
 - 表面上はスライダーで優しいが、
 - 入力しても何も蓄積されない
 - 次を開いたときに何も変わっていない
 - これは「本音を出しても意味がない」という体験を再生産する危険がある。

なので、

UI は良い線いっているのに、バックエンドとの回路がまだ 0 本というのが現状です。

2. 目標：スライダーが「世界を1ステップ更新する」構造にする

ここからは、できるだけ既存の構成を壊さずに、

スライダーを以下 3 つの役割に落とし込みます：

1. **Episode** ログ入力 (Layer A)
2. 「何を大事にしたいか」という **Reward** 設定 (Layer B)
3. その場の **What-if** シミュレーション (Layer C)

そして、**Layer A/B/C** からの値が：

- バックエンドに Episode + Preference として保存される
- 後日の Outcome 入力と結びつく

- ・ 次回のシミュレーションや「おすすめ準備」計算に反映されるというループを作る。

3. 最小の“縦の1本”を決める

いきなり全部やろうとせず、まずは **1 本の縦の流れ** を完成させるのが合理的です。

- ① スライダーを動かす
- ② 「今回のプランを保存」ボタンを押す
- ③ バックエンドに *Scenario + Preparation + Preference* が保存される
- ④ *Episode ID* が返ってくる
- ⑤ 後日、その *Episode ID* に対して *Outcome* を入力する
- ⑥ 「あなたの過去 *N* 回の中で、この準備パターンは平均これだけ効いてきた」が見える

ここだけをまず定義します。

4. API/データフローを具体化する

4.1 フロント → バックエンド : **Episode** 下書き保存

新エンドポイント案

```
POST /api/emotion/episodes/draft
```

リクエスト例 (**Layer A + B + C** の状態をまとめて送る)

```
{
  "scenario_type": "interview",
  "topic": "転職理由",
  "scheduled_at": "2025-12-01T19:00:00",
  "location": "online",

  "pre_state": {
    "pre_anxiety": 7,
    "pre_crying_risk": 8,
    "pre_speech_block_risk": 6
  },

  "preparations_planned": {
    "journaling_10m": 4,
    "three_messages": 5,
    "breathing_4_7_8": 3,
    "roleplay_self_qa": 0,
    "safe_word_plan": 0
  },

  "preference_weights_raw": {
    "relief": 5,
    "expression": 7,
    "relationship": 3
  }
}
```

レスポンス例

```
{
  "episode_id": 1234,
  "normalized_weights": {
    "relief": 0.33,
    "expression": 0.47,
    "relationship": 0.20
  }
}
```

ここで重要なのは：

- ・ スライダーの状態がすべて **1 Episode** 分の「ドラフト」として固まること
- ・ `episode_id` が返ることで、後日の **Outcome** と紐付けられること

4.2 バックエンド実装のざっくり構造

backend/cqox/emotion/service.py にサービス層を切るイメージ：

```
# cqox/emotion/service.py

from . import models, schemas
from sqlalchemy.orm import Session

def create_episode_draft(
    db: Session,
    user_id: int,
    draft: schemas.EpisodeDraftCreate,
) -> schemas.EpisodeDraftRead:
    # 1. EmotionScenario 行を作成 (scenario_type, topic, scheduled_at, location)
    scenario = models.EmotionScenario(
        user_id=user_id,
        scenario_type=draft.scenario_type,
        topic=draft.topic,
        scheduled_at=draft.scheduled_at,
        location=draft.location,
        status="planned",
        pre_anxiety=draft.pre_state.pre_anxiety,
        pre_crying_risk=draft.pre_state.pre_crying_risk,
        pre_speech_block_risk=draft.pre_state.pre_speech_block_risk,
    )
    db.add(scenario)
    db.flush() # episode_id を得る

    # 2. PreparationExecution を複数行 insert
    for key, intensity in draft.preparations_planned.items():
        if intensity and intensity > 0:
            db.add(
                models.EmotionPreparationExecution(
                    scenario_id=scenario.id,
                    template_key=key,
                    planned_intensity=intensity,
                )
            )

    # 3. PreferenceProfile を upsert
    normalized = draft.preference_weights_normalized()
    models.EmotionPreferenceProfile.upsert_for_user(
        db,
        user_id=user_id,
        weights=normalized,
    )

    db.commit()

    return schemas.EpisodeDraftRead(
        episode_id=scenario.id,
        normalized_weights=normalized,
    )
```

4.3 フロントでの「保存」ボタン

今の状態：

- Layer A/B/C のスライダーを動かすと、右側の「予測される結果」バーが変わるだけ
- どこにも POST が飛んでいない

変更案：

- Layer C の下にボタンを追加：

```
<Button onClick={handleSavePlan}>
  今回の準備プランを保存する
```

```
</Button>
```

ロジック (TanStack Query の mutation 例)

```
const saveEpisodeDraftMutation = useMutation({
  mutationFn: async (payload: EpisodeDraftPayload) => {
    const res = await fetch("/api/emotion/episodes/draft", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload),
    });
    if (!res.ok) throw new Error("Failed to save draft");
    return res.json() as Promise<{ episode_id: number }>;
  },
  onSuccess: (data) => {
    setEpisodeId(data.episode_id);
    toast.success("準備プランを保存しました");
  },
});
```

`payload` を構成するときに、

Layer A/B/C の React state を全部まとめる。

5. 「辛い人でも触れる」ための 2 モード設計

本当にしんどい人向けには、「ログに残さずシミュレーションだけ」のモードも必要です。

5.1 UI にモードトグルを追加

- Layer C のあたりにチェックボックス：
 - ☐ 今回はログを残さず、シミュレーションだけ行う
- ON の場合：
 - 保存ボタンを非活性化 or 「保存しないで閉じる」だけにする
 - バックエンドにリクエストを飛ばさない

これで、

- 「とりあえず触ってみたいだけ」の人は何も残さず試せる
- 「今回はちゃんと向き合ってみる」人は保存してデータにする

という二段階の敷居を作れる。

6. 後日の Outcome 入力とループ完結

もう一歩進んで、**Episode リスト → Outcome 入力画面** を足すと、はじめて「循環」ができます。

- トップに「今後の予定」「過去のエピソード」の一覧を出す。
- `status="planned"` で `scheduled_at <= now` のものに「結果を記録する」ボタン。
- そこで再び **スライダーだけで**：
 - `stress_during`
 - `stress_after`
 - `crying_level`
 - `speech_block_level`
 - `expression_score`
 - `relationship_impact`
- `POST /api/emotion/scenarios/{id}/outcomes` へ送る。

→ これを 10~20 回回せば、**個人内の回帰モデル**や

「この準備 × この状態の時に、あなたにはこれくらい効いてきた」という集計が作れる。

7. 実際の開発コマンド (最小)

バックエンド

```
# Emotion ドメインのブランチへ
git checkout -b feature/emotion-episode-draft

cd backend
# スキーマ・モデル追加後
alembic revision -m "add emotion episode draft and preference profile"
alembic upgrade head

pytest tests/emotion -q
uvicorn cqox.api.main:app --reload --port 8000
```

フロントエンド

```
cd ../frontend
npm run lint
npm run test # もし設定済みなら
npm run dev # http://localhost:5173/emotion で UI 確認
```

What this does (今回の設計変更がすること)

- **Goal:**
今の Emotion CQOx UI (Layer A/B/C のスライダー) を、「見た目がきれいなデモ」から、一回一回の操作がユーザーの世界を更新し、次回の提案に効いてくる“ループ”を持ったツールに変える。
- **Inputs:**
 - Layer A: 今のしんどさ / 泣きそう度 / 詰まりそう度
 - Layer B: 楽しさ vs 伝