# CQOx_gen 拡張案

Thought for 1m 19s >

3行サマリ

1. CQOx_gen のアーキテクチャを前提に、「Emotion CQOx（仮称）」のフル仕様書ドラフトを1本の Markdown としてまとめました。 GitHub
2. 中身は「Episode＋Preparation＋Outcome＋Reflection」を核にしたデータモデル・API・UI・分析アルゴリズム・安全ガードレールまで、Claude がそのまま設計/実装分解に使えるレベルで記述しています。
3. 医療行為は行わず、それでも“人の命に触る”前提で、誤用しにくい因果・分析の設計と、リスク検知／専門機関への誘導をプロダクト側に埋め込む方向にしています。

---

以下、そのまま Claude に渡せる仕様書ドラフトです（ファイル名案：`EMOTION_CQOx_SPEC.md`）。

---

## Emotion CQOx – Emotional Episode Optimizer（仕様ドラフト）

### 0. 概要

#### 0.1 プロダクトの一文定義

> *Emotion CQOx は、「面接・1on1・大事な話し合い」などの**感情負荷の高いエピソード**ごとに、*
> *事前準備と結果をログし、**本人にとって“楽で、ちゃんと伝えられる”準備パターン**を見つけるための*
> ***非医療・非診断**の意思決定支援プラットフォームである。*

#### 0.2 CQOx_gen との関係

- ベースアーキテクチャは **CQOx_gen** をそのまま利用する。 GitHub
  - Backend: FastAPI + PostgreSQL + Redis + Celery + S3/MinIO
  - Frontend: React + Vite + TanStack Query
  - Infra: docker-compose / k8s, Prometheus + Grafana + OpenTelemetry
- Emotion CQOx は、既存 CQOx の「マーケ施策」ではなく、
  **“個人 × Episode（出来事）ベースの因果・探索モジュール”** として同じプラットフォーム上に載せる。

#### 0.3 ゴール

1. **個人レベルのゴール**
   - 面接・1on1・大事な話し合いなどで「自分の想いを話そうとすると涙が出る／固まる」人が、
     - どんな条件・話題・準備でそうなりやすいかを可視化し、
     - 「自分に合う準備・言い方・ペース配分」をデータベース化できるようにする。
2. **分析レベルのゴール**
   - 大規模 RCT ではなく、
     - エピソード内比較・本人内比較・シンプルな回帰
       を中心にしながらも、
       **CQOx と同じ「因果の言語」で準備パターンの効果を説明できるレベル**まで持っていく。
3. **安全・倫理レベルのゴール**
   - **医療行為・診断・治療は一切行わない。**
   - 自殺・自傷などのハイリスクケースは、
     - LLM を使わず
     - 専門機関への誘導メッセージ＋緊急連絡先表示
       を行う、「利用を続けるのではなく離脱を促す」設計にする。

#### 0.4 非ゴール（明示的にやらないこと）

- うつ病・PTSD 等の診断・評価・治療効果判定
- 医療専門家の判断を代替するスコアリング
- 自殺リスク予測モデルの自動運用（検知の補助までは許すが、最終判断は常に人間とする）
- 「涙をゼロにする」「感情を消す」ことそのものの最適化
  → 目的関数は常に「**楽さ × 伝達度 × 関係性**」の多目的最適化とする。

# 1. ドメインモデル

## 1.1 抽象モデル

Emotion CQOx の中核は、トピック非依存の抽象モデル：

- **Episode** – 1 回の出来事（面接・1on1・話し合い・登壇など）
- **Preparation / Intervention** – Episode 前に行う準備・介入
- **Outcome** – Episode 後の主観＋客観アウトカム
- **Reflection** – 後日書く振り返り・学び

Emotion 限定ではなく、後に学習・DevOps・マーケにも流用可能な構造にする。

## 1.2 Emotion ドメインの具体エンティティ

### 1.2.1 EmotionScenario（エピソード）

- 想定例：
  - 転職面接で自己紹介・退職理由を話す
  - 上司との評価面談
  - パートナーとの大事な話し合い
- 主要属性：
  - `id`, `user_id`
  - `scenario_type`: `interview`, `one_on_one`, `partner`, `family`, `friend`, `client`, `other`
  - `topic`: `"転職理由"`, `"評価面談"`, `"別れ話"`, など自由入力だがタグ化も視野
  - `scheduled_at`: 予定日時（任意）
  - `location`: `"online"`, `"offline"`, 自由テキスト
  - 事前状態（0–10 の Likert）
    - `pre_anxiety`: 不安度
    - `pre_crying_risk`: 泣きそう度
    - `pre_speech_block_risk`: 言葉が詰まりそう度

### 1.2.2 EmotionPreparationExecution

- 属性：
  - `scenario_id`
  - `template_key`: 準備テンプレート識別子
    - 例：`journaling_10m`, `three_messages`, `breathing_4_7_8`, `roleplay_self_qa`, …
  - `intensity`: 0–10（真面目さ・実施度）
  - `duration_minutes`
  - `notes`: 自由記述

### 1.2.3 EmotionConversationOutcome

- Episode 直後のアウトカム：
  - `stress_during`, `stress_after`（0–10）
  - `crying_level`（0–10）
  - `speech_block_level`（0–10）
  - `expression_score`（0–10、「伝えたいことをどれくらい伝えられたか」）
  - `relationship_impact`（-5〜+5、「関係がどう変わったか」）
  - `partner_reaction`: enum (`very_positive`〜`very_negative`, `unknown`)

### 1.2.4 EmotionReflection

- 後日（任意）の振り返り：
  - `days_after`
  - `would_repeat_preparation`（0–10、「今回の準備をまたやりたいか」）
  - `free_text`

# 2. ユースケース / ユーザーストーリー

## 2.1 代表ユースケース（涙が出てしまう面接）

> *U1:「転職面接で、退職理由を話すときに毎回涙が出そうになる。*
> *自分が異常なのか、どう準備すれば少しでも落ち着いて話せるのか知りたい。」*

- Episode: 転職面接（`scenario_type = "interview"`）
- Preparation:
  - 退職理由の書き出し（`journaling_10m`）
  - 言い方を3パターン用意（`three_messages`）
- Outcome:
  - 実際どのくらい涙が出たか / 話せたか / 相手の反応
- System が返すもの：
  - 「似たエピソード 5 件の中で、
    `journaling_10m` を真面目にやったときは、
    泣きレベルは下がっていないが、**伝達度が明らかに上がっている**」
  - 「逆に、何も準備しないと、泣きレベルは少し下がるが、**伝達度も一緒に落ちている**」
  - →「涙をゼロにする」ではなく、「**泣いてもちゃんと伝えられるパターン**」を提示。

### 2.2 他ユースケース（要約のみ）

- 上司との 1on1 で、評価や不満を落ち着いて伝えたい
- パートナーに、将来やお金の話をしたいが、感情があふれやすい
- 家族に介護・進路などの話を切り出したい
- カンファレンス登壇・LT の前後での緊張とパフォーマンス

---

## 3. システムアーキテクチャ

### 3.1 全体構成

CQOx_gen の構成を踏襲したうえで、Emotion モジュールを追加する。 GitHub

```
graph TB
  subgraph "User Interface"
    EmotionUI[Emotion Console<br/>━━━━━━━━━━<br/>• Episode Log<br/>• Preparation Planner<br/>• Personal Patterns Dashboard]
  end

  subgraph "Application Backend (FastAPI)"
    APICore[Core CQOx API<br/>/api/...]
    APIEmotion[Emotion API<br/>/api/emotion/...]
  end

  subgraph "Analytics & Causal Engine"
    Stats[Within-Subject Statistics<br/>• ΔStress<br/>• ΔExpression]
    Models[Exploration Engine<br/>• Simple Regression<br/>• Contextual Bandit (future)]
  end

  subgraph "Data Infrastructure"
    PG[(PostgreSQL<br/>emotion_* tables)]
    Redis[(Redis Cache)]
    S3[(S3/MinIO<br/>Optionally journaling backups)]
  end

  subgraph "Observability"
    Prom[Prometheus]
    Graf[Grafana]
    OTel[OpenTelemetry]
  end

  EmotionUI --> APIEmotion
  APIEmotion --> PG
  APIEmotion --> Stats
  Stats --> Models
  Models --> APIEmotion

  APIEmotion --> Prom
```

```
  Prom --> Graf
  OTel --> Graf
```

### 3.2 既存コンポーネントの再利用

- 認証・認可：既存の JWT + OAuth2 をそのまま使用。 GitHub
- DB 接続・マイグレーション：既存 Alembic / init_db パイプラインに Emotion 向けテーブルを追加。
- Observability：Emotion エンドポイントにもメトリクス（呼び出し回数、エラー率）を追加し、Grafana ダッシュボードに統合。
- デプロイ：docker-compose / k8s マニフェストに新コンテナは追加しない。既存アプリ内のドメイン追加に留める。

---

## 4. API 設計（Emotion モジュール）

### 4.1 エンドポイント一覧（v1）

- `POST /api/emotion/scenarios`
- `GET /api/emotion/scenarios`
- `GET /api/emotion/scenarios/{id}`
- `POST /api/emotion/scenarios/{id}/preparations`
- `POST /api/emotion/scenarios/{id}/outcomes`
- `POST /api/emotion/scenarios/{id}/reflections`
- `GET /api/emotion/dashboard/summary`
  - per scenario_type × preparation_template の集約
- `GET /api/emotion/safety/check`
  - 入力テキストにハイリスク表現が含まれるかどうかの軽量チェック

### 4.2 例：Scenario 作成 API（スキーマ要約）

```
// EmotionScenarioCreate (request)
{
  scenario_type: "interview" | "one_on_one" | "partner" | "family" | "friend" | "client" | "other",
  topic: string,
  scheduled_at?: string, // ISO8601
  location?: string,
  pre_anxiety?: number,           // 0–10
  pre_crying_risk?: number,       // 0–10
  pre_speech_block_risk?: number  // 0–10
}

// EmotionScenarioRead (response)
{
  id: number,
  ... // 上記に加え、created_at / updated_at など
}
```

### 4.3 ダッシュボード Summary API の出力イメージ

```
{
  "by_preparation": [
    {
      "template_key": "journaling_10m",
      "scenario_type": "interview",
      "n_with": 8,
      "n_without": 5,
      "delta_stress": {
        "mean": 1.8,
        "ci95": [0.3, 3.1]
      },
      "delta_expression": {
        "mean": 2.5,
        "ci95": [1.0, 3.9]
      },
      "confidence_label": "medium"
```

```
    },
    ...
  ]
}
```

---

## 5. 分析・因果モデリング設計

### 5.1 データの前提

- 各ユーザのログは、N が数十〜数百エピソード程度を想定。
- 介入（準備）はランダムではなく、ユーザが自分で選ぶ → **観察データ**。
- データは「本人内比較」が主で、クロスユーザ集約は **統計的サマリの提示まで**（診断・スコアリングはしない）。

### 5.2 MVP の分析ロジック

#### 5.2.1 単純差分（pre vs post）

- Episode ごとに「楽さ指標」を定義：
  - 例：
    Y_i = \text{pre_anxiety}_i - \text{stress_after}_i
  - 表現度：
    $Z_i = \text{expression\_score}_i$
- 準備パターン $p$ ごとに：
  - `with`：その準備をした Episode 群
  - `without`：しなかった Episode 群
- 効果量：
  - ΔY_p = 平均(Y_with) − 平均(Y_without)
  - ΔZ_p = 平均(Z_with) − 平均(Z_without)
- サンプル数と分散から、t-検定ベースの 95% CI を計算し、「low / medium / high confidence」にラベル化。

#### 5.2.2 共変量を入れた線形回帰（v1.5〜）

- 説明変数：
  - Episode の種類（interview / partner / ...）
  - 時刻帯（朝/昼/夜）
  - 相手との関係（上司・同僚・家族）
- 目的変数：
  - Y（楽さ）と Z（伝達度）
- モデル：
  - 単純な線形回帰 or Elastic Net
  - ユーザごとに推定し、係数の方向と大きさだけを UI に反映。

#### 5.2.3 将来：コンテキスト付きバンディット

- エピソード開始前に：
  - context: episode_type, topic, pre_anxiety, pre_crying_risk など
- 準備パターン＝Bandit のアーム
- 報酬 = α * ΔY + β * ΔZ（係数はユーザが調整可）
- LinUCB / Thompson Sampling で、「**今のあなたの状態なら、この準備が一番期待値高そう**」を提示。

### 5.3 因果の扱い方（CQOx との整合）

- マーケ CQOx と違い、完全な因果推定は難しい前提に立つ：
  - 自己選択バイアス（しんどいときほど準備する など）
  - エピソードごとに confounder を完全に観測できない
- それでも CQOx の世界観に寄せるために：
  - 「反実仮想（準備しなかった世界）」を推定していることを明示
  - 推定結果には常に「不確実性（CI・confidence ラベル）」をセット
  - ダッシュボードでは必ず「**これは“あなたの過去データに基づく推定”であり、確定ではない**」と表示

---

## 6. UX / 画面設計（概要）

### 6.1 Emotion Console（トップ）

- コンポーネント：
  - 直近 Episode のタイムライン
  - 「次の大事なエピソード」のカード（面接日など）
  - 「自分に効きやすい準備 TOP3」（scenario_type 別）

### 6.2 Episode 登録画面

- ステップ形式：
  1. 基本情報（種類・相手・日時）
  2. 話題（例：退職理由、評価、将来、お金、関係性）
  3. 事前状態（不安度・泣きそう度・言葉が詰まりそう度）
  4. 準備プラン選択（テンプレ＋自分の追加）

### 6.3 Outcome 入力画面（当日〜直後）

- 質問項目：
  - 面接中のつらさ
  - 終わった後の楽さ
  - 涙・言葉の詰まり
  - 伝えられた度
  - 相手の反応

### 6.4 Reflection 画面（後日）

- テキストボックス＋「またこの準備をしたいか」スライダー。
- ハイリスクキーワード検知（次節）と連動。

### 6.5 Safety 挙動（UIレベル）

- Emotion タブのトップと各フッターに、
  - 「医療・診断の代わりにはならない」
  - 「強い落ち込みや自傷の考えがある場合は専門機関へ」
    を常に明示。
- ハイリスク表現を検知した場合：
  - 入力を**そのまま LLM に投げない**
  - 代わりに、
    - 専門窓口の一覧
    - 緊急時の行動（119 など国ごとのもの）
      を表示し、利用継続より安全確保を優先する。

---

## 7. 安全・倫理設計

### 7.1 境界条件

- システム / ドキュメント / UI のすべてで、以下を徹底：
  - 「診断しない」
  - 「症状名を自動で付与しない」
  - 「治療プランを提示しない」
- Emotion CQOx は「**自分にとって楽で伝えやすい準備パターンを探すためのログ＋可視化ツール**」に限定。

### 7.2 ハイリスクテキストの取り扱い

- 対象：
  - 自殺・自傷・他害に関する明示的表現
  - 「消えたい」「死にたい」などの強い希死念慮表現
- 実装：

    - 軽量なルールベース＋（将来）専用小モデルで検知
    - 検知したテキストは
        - LLM への送信禁止
        - 可能なら暗号化・短期保持（or 即削除）ポリシー
- UI：
    - 「この内容は Emotion CQOx では扱えません」と明示し、
        - 医療機関・相談窓口一覧へのリンク
        - 危機介入の基本手順
          を提示。

---

## 8. 実装フェーズとマイルストーン

### 8.1 フェーズ 0 – モデル・スキーマ確定

- タスク：
    - Emotion ドメインの ORM モデル・Pydantic スキーマ定義
    - Alembic マイグレーション作成
    - `/api/emotion/scenarios` の CRUD 実装
- 成果：
    - CLI から Episode を登録・取得できる状態。

### 8.2 フェーズ 1 – ログ＋簡易ダッシュボード

- タスク：
    - Preparation/Outcome/Reflection エンドポイント追加
    - Summary API（単純差分＋CI）
    - フロントエンドに Emotion タブ追加
- 成果：
    - 自分の Episode と準備・結果を見られる
    - 準備パターンごとの ΔStress / ΔExpression がバー＋テーブルで可視化。

### 8.3 フェーズ 2 – 安全機能＆説明レイヤ

- タスク：
    - ハイリスクテキスト検知
    - 「なぜ涙が出るのか？」などの**短いメカニズム解説**コンポーネント追加
    - UI 上のディスクレーマー整備

### 8.4 フェーズ 3 – 探索アルゴリズム（**Optional**）

- タスク：
    - ユーザごとの簡易線形モデル
    - コンテキスト付きバンディットの PoC
- 成果：
    - Episode 作成時に「おすすめ準備プラン」の提示が可能になる。

---

## 9. 実装用メモ（CQOx_gen への統合のための最小コマンド）

> *※ ここは開発者向け。Claude がコード生成する際のガイドになるように書く。*

### 9.1 バックエンド側の追加

```
# 1. Emotion ドメイン用ディレクトリ
cd backend
mkdir -p cqox/emotion
touch cqox/emotion/__init__.py
touch cqox/emotion/models.py
touch cqox/emotion/schemas.py
```

```
# 2. Emotion API ルータ
touch cqox/api/emotion.py

# 3. マイグレーション（Alembic 前提）
alembic revision -m "add emotion domain tables"
# -> models を参照するように upgrade() を実装
alembic upgrade head
```

### 9.2 フロントエンド側の追加

```
cd frontend
# Emotion タブ用ページ
mkdir -p src/pages/emotion
touch src/pages/emotion/EmotionDashboard.tsx
touch src/pages/emotion/EpisodeForm.tsx
touch src/pages/emotion/OutcomeForm.tsx

# ルーティングに /emotion を追加
# src/router/index.ts などに Route 追加
```

### 9.3 テスト / 検証

```
# Backend 単体テスト
cd backend
pytest tests/emotion  # 追加予定

# Frontend
cd ../frontend
npm test
npm run lint
npm run dev  # 手動 E2E チェック

# フルスタック（既存 Quick Start と同様）
cd ..
docker-compose up --build
# http://localhost:3001/emotion で UI を確認
```

---

## What this does（この仕様書がすること）

- **Goal:** CQOx_gen 上に「Emotion CQOx」モジュールを追加するための、アーキテクチャ・データモデル・API・UX・分析・倫理の一貫した仕様を定義する。
- **Inputs:** 既存 CQOx_gen の構成（FastAPI + React + PostgreSQL + Celery + Grafana など）、そして「大事な話をすると涙が出てしまう」といった具体的なユーザー体験。 GitHub
- **Outputs / side-effects:**
  - Claude / LLM がこの仕様を読み取り、
    - 具体的な ORM / Pydantic / FastAPI コード
    - React コンポーネント
    - マイグレーション

    を自動生成・分解できるレベルの情報を提供する。
  - 医療境界と倫理ガードレールが、最初の設計段階から明示的に組み込まれる。
- **Why this approach:**
  - CQOx と同じ「因果・不確実性・意思決定」の言語に揃えつつ、
    人の感情と命に関わる領域で「やってはいけないこと」を最初から仕様に埋め込むため。
- **How to run / verify:**
  - 上記 9. のコマンドで Emotion ドメインの最小実装を生成し、
    - API が 200 を返すこと
    - Episode → Preparation → Outcome → Summary の一連のフローが UI で動くこと

    を E2E で確認する。
  - その後、ログを数件入れて ΔStress / ΔExpression の算出が期待通りかを手計算で検証する。

---

### Expert insight (Google/Meta/NASA level)

**感情・メンタル領域のツールでは、「何を最適化するか」を最初に多目的で定義しておくことが致命的に重要**です。

- 「泣かない」を単一目的にすると、人は簡単に「感情を押し殺す」方向に最適化され、
  長期的には燃え尽きや離職・破綻リスクをむしろ高めかねません。
- 最初から「楽さ × 伝達度 × 関係性」のように複数軸を持ち、
  それらのトレードオフを可視化する設計にすると、
  **人間の複雑な目的関数を尊重したまま、“扱い方を増やす”方向のプロダクト**になります。