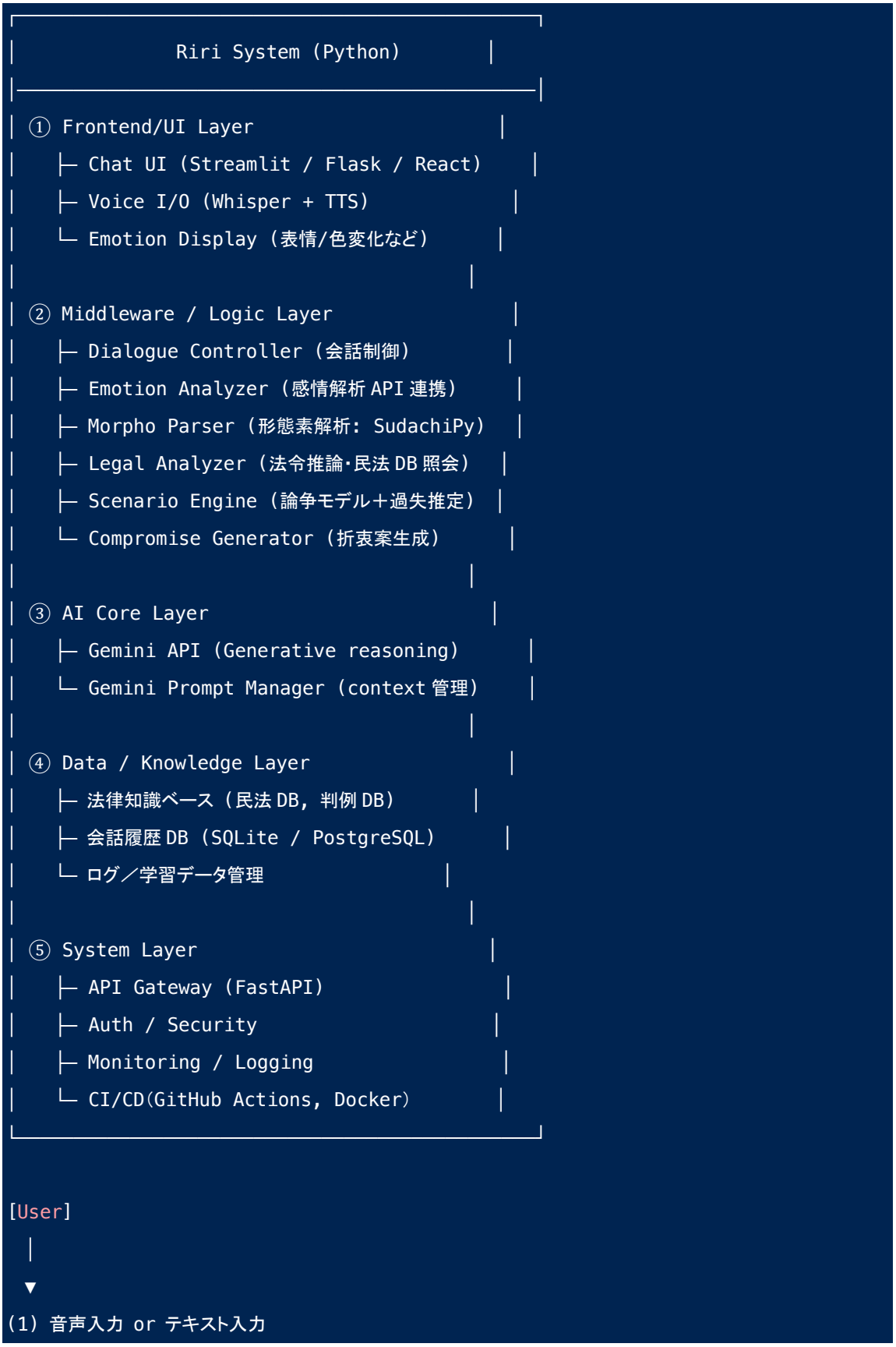
Riri System（Python）
① Frontend/UI Layer
├ Chat UI（Streamlit / Flask / React）
├ Voice I/O（Whisper + TTS）
└ Emotion Display（表情/色変化など）
② Middleware / Logic Layer
├ Dialogue Controller（会話制御）
├ Emotion Analyzer（感情解析 API 連携）
├ Morpho Parser（形態素解析: SudachiPy）
├ Legal Analyzer（法令推論・民法 DB 照会）
├ Scenario Engine（論争モデル＋過失推定）
└ Compromise Generator（折衷案生成）
③ AI Core Layer
├ Gemini API（Generative reasoning）
└ Gemini Prompt Manager（context 管理）
④ Data / Knowledge Layer
├ 法律知識ベース（民法 DB, 判例 DB）
├ 会話履歴 DB（SQLite / PostgreSQL）
└ ログ／学習データ管理
⑤ System Layer
├ API Gateway（FastAPI）
├ Auth / Security
├ Monitoring / Logging
└ CI/CD(GitHub Actions, Docker)
[User]
▼
(1) 音声入力 or テキスト入力

```
  │
  ▼
[Frontend/UI Layer]
  └→ 音声→(Whisper)→テキスト化
  └→ テキスト→Emotion Analyzer へ送信
  │
  ▼
[Middleware]
  ├─ 形態素解析（SudachiPy）
  ├─ 感情分析（感情 API）
  ├─ Gemini へプロンプト構築（Prompt Manager）
  │
  ▼
[Gemini Core]
  ├─ 発話内容・感情・法的観点を統合的に判断
  ├─ 折衷案 / 過失割合 / 和解文案を生成
  ▼
[Middleware]
  └─ Gemini 出力を再構成(Compromise Generator)
  └─ 結果を Emotion Display へ
  │
  ▼
[Frontend]
  ├─ テキスト出力
  ├─ 音声合成（TTS）
  └─ 表情・UI 表示更新
  │
  ▼
[User]
```

項目	OpenAI 中心構成との差異	理由

モデル統合	Gemini 1.5 Pro などを中心に置き、感情・推論・文脈処理を統合	Gemini はマルチモーダル対応が強く、法的要約・感情補正が自然
プロンプト制御	Gemini Prompt Manager を独立化	プロンプト階層（法的文脈・感情・過失判定）を柔軟に扱うため
外部 API 利用	感情解析、TTS、STT、形態素解析は別モジュールで補完	Gemini 単体では高精度形態素解析・音声認識が限定的
データ管理	Gemini API トークンとユーザー履歴を統合管理	会話履歴を元に過失学習モデルへ発展させる余地あり

分類	技術
Web Framework	FastAPI / Flask
フロント	Streamlit / React + Tailwind
AI Core	Google Gemini API
STT	Whisper / Google Cloud Speech-to-Text
TTS	Google Cloud Text-to-Speech
形態素解析	SudachiPy / Janome
DB	SQLite（開発）→ PostgreSQL（運用）
Logging	Loguru / ELK Stack
Deploy	Docker / Cloud Run / GCP
Auth	Firebase Auth / OAuth 2.0

下記の記述は OpenAI 対応型

全ての Python サービスは FastAPI（軽量で OpenAPI 対応）で作成し、Pydantic モデルで入力検証を統一。

非同期処理／長時間処理は Celery（または RQ）+ Redis でバックグラウンド化。

コンテナ化（Docker）→ Kubernetes（EKS/GKE/AKS）で運用。CI/CD は GitHub Actions / GitLab CI。

モジュール構成と責任（詳細）

各モジュール名・責任・主要ライブラリ／インターフェースを列挙します。

A. API Gateway（FastAPI）

役割：外部と内部サービスのルーティング、認可チェック、レート制御、レスポンス整形。

実装：FastAPI + uvicorn + Gunicorn。Ingress の後に配置。

ミドルウェア：Auth（JWT 検証）、Request ID、OpenTelemetry tracing。

B. Auth Service

役割：ログイン、OAuth2 フロー、RBAC、トークン発行・リフレッシュ。

実装：FastAPI、PyJWT、Keycloak 連携可。

出力：アクセストークン（JWT）、ユーザーロール。

C. Dialog Manager（会話制御）

役割：セッション管理、対話ステートマシン、会話履歴の取りまとめ、呼び出し順序制御。

実装：FastAPI、Redis（セッション・スロット管理）。

API: /session/{id}/message を受け、NLU→Analysis→Proposal のオーケストレーション。

D. NLU Service（形態素解析・NER・感情）

役割：形態素解析（MeCab / SudachiPy）、NER（法律固有語辞書）、意図分類、感情スコアリング。

実装：FastAPI + transformers（cl-tohoku/bert-japanese） / GiNZA。辞書は DBor ファイルで読み込み。

出力：tokens, entities, intent, emotion_scores, confidence。

E. Media Service(ASR/TTS Gateway)

役割:音声ファイルの受取り→ASR(Whisper/Google)→テキスト、Gemini 出力→TTS で音声生成。

実装:FastAPI でアップロードエンドポイント、Celery で長時間処理(音声ファイル変換等)。

入出力:audio file (S3 URL) ⇄ transcript / generated audio.

F. Legal KB & Search Layer

役割:法令 DB(Postgres + Elasticsearch)と判例の全文検索、メタデータ管理、ベクトル検索 for 類似判例(Milvus)。

実装:ETL バッチで e-Gov 同期、全文は Elasticsearch、類似検索はベクトル DB。

出力:matched laws/cases + relevance scores + text snippets。

G. Analysis Service(過失割合推定等)

役割:論点→数値化→統計/機械学習モデルで過失割合推定。SHAP 等で説明可能性出力。

実装:FastAPI + scikit-learn / PyMC3(ベイズ) / PyTorch(学習系)。モデルは MLflow で管理。

出力:fault_estimates: party→ratio, confidence, reasoning_features。

H. Proposal Service(折衷案生成)

役割:Gemini を呼んで中立的な和解案を生成。テンプレート適用、感情トーン調整、倫理フィルタ通過。

実装:FastAPI、Gemini HTTP client(python-requests or httpx)、出力ポストプロセス(filter & cite)。

出力:proposal_text, actionable_items, legal_references, tone_metadata。

I. Governance / Ethics Filter

```
役割：性別・差別表現の検出、出力のバイアス検査、出力ブロッキングと代替文言化。

実装：ルールベース + ML（バイアス検出スコア）、管理 UI でルール編集。

J. Storage & Logging

Postgres（核心データ：users, sessions, messages, proposals, legal_refs）

Redis（cache, session）

S3（audio, backups）

Elasticsearch（logs + law text）

OpenTelemetry / ELK for logs & traces.
```