┌──────────────────────────────────────────────┐

│ 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.