**1. Análise do Progresso nos Últimos Dias (01/05/2025 a 03/05/2025))**

Os arquivos fornecidos, especialmente o documento 01-05-2025 a 03-05-2025.docx,, detalham as atividades realizadas nos dias 01/05/2025, 02/05/2025 e 03/05/2025. Vou resumir o progresso, conectar com nossas interações de hoje, e identificar pendências.

**01/05/2025**

**Resumo do Documento**::

* **Persistência da Sessão**: :
  + Implementada funcionalidade para salvar e recuperar o session\_id usando localStorage..
  + Ajustados main.js (agora chat.js)) e state.js para chamar loadSessionData e buscar histórico do backend ao carregar a página.
* **Backend**: :
  + Corrigido o endpoint //api/session-history/:session\_id para suportar sessões anônimas.
  + Ajustado //api/call-bot para salvar histórico de sessões anônimas com user\_id como 'anon'..
* **Correções de Erros**: :
  + Resolvido erro de sintaxe em server.js..
  + Corrigidos erros 404 ao carregar histórico de sessões anônimas.
* **Refatoração do Frontend**: :
  + Movidas funções de formatação de histórico de main.js para logs.js..
  + Movida função de exibição da resposta final para ui.js..
* **Melhoria Visual**: :
  + Corrigido problema de texto longo em logs com CSS ((white-space: pre-wrap; word-wrap: break-word)) em styles.css..
* **Sincronização**: :
  + Atualizado o repositório no GitHub.
* **Próximos Passos Definidos**: :
  + Prioridade 1: Implementar autenticação (login, registro, gestão de token JWT).
  + Prioridade 2: Gerenciar múltiplas conversas (listar e selecionar sessões anteriores).

**Status**:: Todas as tarefas listadas foram concluídas, com progresso significativo na persistência de sessões e correções de erros. A autenticação foi parcialmente abordada em 02/05/2025, e o gerenciamento de múltiplas conversas começou em 03/05/2025 com client-sessions.html..

**02/05/2025**

**Resumo do Documento**::

* **Planejamento**: :
  + Priorizados papéis de usuário (Master/Assistente) e gestão de clientes/vinculação do chat, adiando dashboard e lista genérica de sessões.
* **Papéis de Usuário (Backend)**: **)**:
  + Adicionadas colunas role (padrão 'master')) e master\_user\_id (nulo) à tabela users em egide.db..
  + Ajustado //api/login para incluir role no token JWT.
* **Gestão de Assistentes (Backend)**: **)**:
  + Criado endpoint POST /api/assistants para criar contas de auxiliares vinculadas a um Master.
  + Criado endpoint GET /api/assistants para listar assistentes de um Master.
* **Gestão de Assistentes (Frontend)**: **)**:
  + Criada página assistants.html com formulário e tabela para assistentes.
  + Criado manage-users.js para interagir com endpoints //api/assistants..
  + Adicionado link "Gerenciar Assistentes" em index.html..
  + Corrigidos erros de importação/exportação e URLs de API.
* **Gestão de Clientes (Backend)**: **)**:
  + Criada tabela clients em egide.db com colunas client\_id, , user\_id, , name, , cpf, , dob, , created\_at..
  + Criados endpoints GET /api/clients e POST /api/clients para listar e cadastrar clientes.
* **Gestão de Clientes (Frontend)**: **)**:
  + Criada página clients.html..
  + Criado clients.js para listar e adicionar clientes.
  + Corrigidos erros de importação ((createAuthHeaders, , API\_CLIENTS\_URL).).
* **Próximos Passos Definidos**: :
  + Vincular sessões a clientes (criar tabela sessions,, ajustar //api/call-bot).).
  + Ajustar frontend ((clients.js, , main.js, , api.js)) para iniciar chat a partir de um cliente, enviando client\_id e usando sessionStorage para contexto.
  + Testar fluxo completo de vinculação cliente-sessão.

**Status**:: Todas as tarefas foram concluídas. A vinculação cliente-sessão foi implementada em 03/05/2025, mas o teste completo do fluxo ainda está pendente.

**03/05/2025 (Antes das Nossas Interações))**

**Resumo do Documento**::

* **Retomada e Análise Inicial**: :
  + Revisado o estado do projeto com base em objetivo.txt, , arquitetura.txt, , backend/package.json, , frontend/index.html..
* **Análise do Backend ((server.js)**: **)**:
  + Compreendido o funcionamento de middlewares, conexão com SQLite, autenticação, gestão de clientes/assistentes, chamadas aos bots e recuperação de histórico.
* **Vinculação Sessão-Cliente (Backend)**: **)**:
  + Criada tabela sessions em egide.db para relacionar session\_id, , user\_id, , client\_id..
  + Modificado //api/call-bot para:
    - Receber client\_id..
    - Validar client\_id (pertence ao usuário).
    - Inserir/atualizar registros na tabela sessions..
* **Correção de Erro (Frontend)**: **)**:
  + Resolvido ReferenceError: SESSION\_ID\_STORAGE\_KEY is not defined em api.js importando a constante de state.js..
* **Teste**: :
  + Confirmado (via logs do backend) que a vinculação na tabela sessions funciona.
* **Reestruturação da Navegação (Frontend)**: **)**:
  + Renomeado index.html para chat.html e main.js para chat.js..
  + Criado novo index.html (dashboard básico) e index-page.js com autenticação, logout e controle de acesso por role..
  + Criada página client-sessions.html e client-sessions.js para listar histórico de sessões por cliente.
  + Modificado clients.html//clients.js para incluir link para client-sessions.html..
  + Ajustado handleLoadSession em client-sessions.js para redirecionar para chat.html..
  + Adicionado botão "Voltar" em chat.html//chat.js usando sessionStorage ( (cameFrom).).
* **Correções de Erros**: :
  + Resolvido ReferenceError: sessionId is not defined com event delegation em client-sessions.js..
  + Corrigida falha de renderização em client-sessions.html (loop forEach).).
  + Resolvido SyntaxError: Unexpected end of input (falta de }} em client-sessions.js).).
* **Próximas Etapas Definidas**: :
  + Teste completo do fluxo "Login -> Gerenciar Clientes -> Iniciar Análise -> Chat -> Recarregar Página".
  + Implementar gerenciamento de múltiplas sessões (UI).
  + Refinar papéis/permissões.
  + Construir dashboard completo.

**Status**:: Grande progresso na vinculação cliente-sessão e reestruturação da navegação. O teste completo e o dashboard completo são as principais pendências.

**2. O Que Fizemos Hoje (03/05/2025)**

2. O Que Fizemos Hoje (03/05/2025**)**

Nossas interações hoje focaram em resolver problemas críticos no backend e no frontend, garantindo que o fluxo principal de análise jurídica funcionasse corretamente. Aqui está o que foi feito, com base nas interações e no documento 01-05-2025 a 03-05-2025.docx::

**Resumo das Atividades de Hoje**

1. **Correção de Erros no Backend**: :
   * **Problema**:: O endpoint //api/call-bot retornava SqliteError: no such column: attendance\_id ao tentar registrar uma sessão na tabela sessions..
   * **Causa**:: A coluna attendance\_id não existia no banco de dados egide.db,, embora estivesse definida no esquema em db.js..
   * **Solução**: :
     + Verificado que a coluna attendance\_id já existia na tabela sessions (tentativa de adicionar via ALTER TABLE resultou em duplicate column name).).
     + Corrigida a query de validação do atendimento em bots.js: :

javascript

Copiar

Copiar

*// De:*

const attendanceCheck = db.prepare('SELECT attendance\_id FROM attendances WHERE attendance\_id = ? AND client\_id = ?').get(attendance\_id, client\_id);

*// Para:*

const attendanceCheck = db.prepare('SELECT id FROM attendances WHERE id = ? AND client\_id = ?').get(attendance\_id, client\_id);

* + - Confirmado que o esquema do banco de dados estava correto ((attendance\_id presente em sessions, , id como chave primária em attendances).).
  + **Resultado**:: O endpoint //api/call-bot agora registra sessões corretamente, vinculando session\_id, , user\_id, , client\_id,, e attendance\_id na tabela sessions..

1. **Correção de Erros no Frontend**: :
   * **Problema**:: A chamada à API //api/call-bot falhava devido a uma ordem incorreta de parâmetros na função callBotAPI em chat.js (parâmetro retries recebia attendance\_id).).
   * **Causa**:: Erro na chamada em mainCaseFlow: :

javascript

Copiar

Copiar

technicalReport = await callBotAPI("redator", initialInput, null, user\_id, clientIdToLink, attendanceId);

* + **Solução**: :
    - Corrigida a ordem dos parâmetros em chat.js: :

javascript

Copiar

Copiar

technicalReport = await callBotAPI("redator", initialInput, null, clientIdToLink, attendanceId);

* + - Adicionado log para depuração:

javascript

Copiar

Copiar

console.log('Chamando callBotAPI para redator:', { initialInput, user\_id, clientIdToLink, attendanceId });

* + **Resultado**:: A chamada à API //api/call-bot foi bem-sucedida, permitindo o fluxo completo de análise jurídica (Redator → Médico → Estrategista → Supervisor).

1. **Validação do Fluxo Completo**: :
   * **Teste Realizado**: :
     + Iniciada uma análise para o cliente "PAULO HENRIQUE GONÇALVES" com o caso "dor lombar baixa CID-10 M54, incapacidade desde jan/2024".
     + O fluxo foi executado com sucesso, gerando:
       - Relatório técnico do Redator (Log 3).
       - Análise médica do Médico (Log 5).
       - Análise estratégica do Estrategista (Log 7).
       - Relatório final do Redator (Log 9).
       - Resposta final consolidada do Supervisor (Log 11).
   * **Logs do Backend**: :
     + Confirmada a vinculação correta do client\_id ( (client\_1746308221062\_6csvs)) e attendance\_id ( (attendance\_1746313026885\_5d4no)) na tabela sessions..
     + Resposta bem-sucedida da OpenAI (status 200).
   * **Logs do Frontend**: :
     + Todos os logs foram exibidos corretamente em ##logsIndividuais,, com interações do usuário e respostas dos bots.
   * **Resultado**:: O fluxo principal de análise jurídica está funcionando perfeitamente, com integração completa entre frontend, backend, banco de dados, e API da OpenAI.
2. **Outras Atividades Mencionadas no Documento**: :
   * **Correções em client-sessions.js**: :
     + Resolvido SyntaxError (falta de catch)) e TypeError ( (escapeHtml com valores null)) em fetchAndRenderClientSessions..
     + Essas correções foram realizadas antes das nossas interações, mas mencionadas no documento.
   * **Dashboard Básico**: :
     + Criado index.html como dashboard inicial com links de navegação e index-page.js para autenticação/logout.
     + Não implementado o dashboard proposto em React (artefato ID: 9da3bba6-df34-4f29-89e7-25de268eba92).
   * **Teste Parcial**: :
     + Confirmada a vinculação cliente-sessão via logs do backend, mas o teste completo (incluindo recarregamento da página) não foi mencionado.

**Pendências Identificadas Hoje**::

* **Teste Completo do Fluxo**: :
  + Ainda não foi confirmado se o teste de recarregar chat.html restaura a sessão vinculada ao cliente corretamente.
* **Dashboard Completo**: :
  + O dashboard atual é básico (links de navegação). A versão em React com seções detalhadas (notificações, clientes, casos) está pendente.
* **Gerenciamento de Múltiplas Sessões**: :
  + A UI para listar e selecionar sessões anteriores (integrada a client-sessions.html ou index.html)) não foi implementada.
* **Refinamento de Papéis/Permissões**: :
  + Não há controle de acesso granular baseado em role (Master vs. Auxiliar).
* **Funcionalidades Futuras**: :
  + Gestão de documentos, prazos, editor de peças jurídicas, OCR, e transcrição de áudios estão planejadas, mas não iniciadas.

**3. Arquitetura Técnica do Sistema (03/05/2025))**

A seguir, detalho a arquitetura técnica do sistema **Égide Jurídico** na data de hoje, incluindo o que está funcionando, os fluxos implementados, os arquivos relevantes, e uma visão geral para referência futura.

**Visão Geral**

* **Tipo**:: Aplicação web moderna com arquitetura cliente-servidor.
* **Objetivo**:: Automatizar e otimizar a análise jurídica, com foco inicial em Direito Previdenciário, através de uma plataforma integrada que utiliza IA (OpenAI GPT-4o) para gerar relatórios e análises.
* **Público-Alvo**:: Advogados (usuários Master) e suas equipes (Auxiliares), com interface intuitiva acessível a usuários com mínima capacitação técnica.
* **Estado Atual**:: Protótipo funcional com fluxo principal de análise jurídica completo, autenticação básica, gestão de clientes/assistentes, e vinculação cliente-sessão. Funcionalidades como upload de documentos, OCR, e editor de peças estão planejadas, mas não implementadas.

**Componentes da Arquitetura**

1. **Frontend**: :
   * **Responsabilidade**:: Interface do usuário, captura de inputs, exibição de dados, interação com o backend via API.
   * **Tecnologias**: :
     + HTML, CSS, JavaScript (ES6+).
     + Vite como ferramenta de build e servidor de desenvolvimento (substituiu http-server).).
     + Bootstrap para estilos (referenciado em styles.css).).
     + i18next para internacionalização ((i18n.js).).
   * **Estrutura de Arquivos** (em frontend/): /):
     + **Páginas**:

Páginas:

* + - * index.html:: Dashboard básico com links de navegação (Clientes, Assistentes, Logout).
      * chat.html:: Interface principal para análise jurídica (interação com bots).
      * clients.html:: Página para listar e cadastrar clientes.
      * client-sessions.html:: Página para listar histórico de sessões por cliente.
      * assistants.html:: Página para gerenciar assistentes.
      * login.html, , register.html:: Páginas para autenticação (implementação básica).
    - **Scripts JavaScript** (em frontend/js/): /):
      * index-page.js:: Lógica do dashboard ((index.html),), incluindo autenticação e logout.
      * chat.js:: Lógica principal da análise jurídica, gerencia mainCaseFlow e interações com bots.
      * clients.js:: Gerencia listagem e cadastro de clientes.
      * client-sessions.js:: Gerencia histórico de sessões por cliente.
      * manage-users.js:: Gerencia assistentes (criação/listagem).
      * api.js:: Funções para chamadas HTTP aos endpoints do backend (ex.: callBotAPI, , getSessionHistory).).
      * ui.js:: Funções de interface (ex.: showProgress, , showError, , showFinalResponse).).
      * logs.js:: Funções para formatação e renderização de logs.
      * state.js:: Gerencia estado global ((AppState, , SESSION\_ID\_STORAGE\_KEY).).
      * dom-elements.js:: Define elementos do DOM ((DOM).).
      * i18n.js:: Configura internacionalização.
      * pdf.js:: Gera PDFs a partir de logs ((downloadConversationAsPdf).).
      * theme.js:: Gerencia temas visuais (claro/escuro).
      * config.js:: Define URLs de API e configurações (ex.: API\_CALL\_BOT\_URL).).
      * auth.js:: Lógica de autenticação (login/logout).
    - **Estilos**:

Estilos:

* + - * frontend/css/styles.css:: Estilos globais, incluindo correção para texto longo ((white-space: pre-wrap).).
  + **Funcionalidades Implementadas**: :
    - Interface para análise jurídica ((chat.html): ):
      * Campo de entrada para descrição do caso.
      * Exibição de logs em ##logsIndividuais (usuário, Redator, Médico, Estrategista, Supervisor).
      * Botão para exportar logs como PDF.
      * Botão "Voltar" usando sessionStorage ( (cameFrom).).
    - Gestão de clientes ((clients.html): ):
      * Listagem e cadastro de clientes via GET /api/clients e POST /api/clients..
      * Link para iniciar análise ((client-sessions.html).).
    - Histórico de sessões ((client-sessions.html): ):
      * Listagem de sessões por cliente.
      * Redirecionamento para chat.html ao selecionar uma sessão.
    - Gestão de assistentes ((assistants.html): ):
      * Cadastro e listagem de assistentes via GET /api/assistants e POST /api/assistants..
    - Autenticação:
      * Login/logout básico via login.html, , register.html,, e auth.js..
      * Redirecionamento para login.html se o token estiver ausente.
    - Indicador de progresso ((showProgress, , clearProgress)) para chamadas à API.
    - Internacionalização ((i18n.js)) com suporte a português ((pt)) e inglês ((en).).
  + **Fluxos Funcionando**: :
    - **Fluxo de Análise Jurídica**: :
      * Usuário faz login ((login.html).).
      * Acessa clients.html,, seleciona um cliente, e clica em "Iniciar Análise".
      * Redirecionado para client-sessions.html,, onde pode iniciar uma nova sessão ou carregar uma existente.
      * Em chat.html,, insere a descrição do caso (ex.: "dor lombar baixa CID-10 M54").
      * A função mainCaseFlow em chat.js orquestra chamadas a callBotAPI ( (api.js),), enviando requisições para //api/call-bot para os bots (Redator, Médico, Estrategista, Supervisor).
      * As respostas são exibidas em ##logsIndividuais e salvas no histórico ((session\_history).).
      * O usuário pode exportar os logs como PDF ou recarregar a página para restaurar a sessão.
    - **Fluxo de Gestão de Clientes**: :
      * Usuário acessa clients.html após login.
      * Lista clientes existentes ((GET /api/clients)) ou cadastra novos ((POST /api/clients).).
      * Clica em "Iniciar Análise" para um cliente, redirecionando para client-sessions.html..
    - **Fluxo de Gestão de Assistentes**: :
      * Usuário Master acessa assistants.html..
      * Lista assistentes ((GET /api/assistants)) ou cadastra novos ((POST /api/assistants).).
    - **Fluxo de Autenticação**: :
      * Usuário faz login em login.html,, recebendo um token JWT.
      * Token é armazenado no localStorage e usado em chamadas à API.
      * Logout remove o token e redireciona para login.html..

1. **Backend**: :
   * **Responsabilidade**:: Lógica de negócios, autenticação, acesso ao banco de dados, integração com a API da OpenAI.
   * **Tecnologias**: :
     + Node.js com Express ((server.js).).
     + SQLite com better-sqlite3 ( (db.js).).
     + JWT para autenticação ((jsonwebtoken).).
     + Axios para chamadas à API da OpenAI.
     + Dotenv para variáveis de ambiente (.(.env).).
   * **Estrutura de Arquivos** (em backend/): /):
     + server.js:: Configura o servidor Express, middlewares (CORS, JSON), rotas, e arquivos estáticos do frontend.
     + db.js:: Define o esquema do banco de dados SQLite ((egide.db)) e cria tabelas ((users, , clients, , attendances, , sessions, , session\_history).).
     + routes/bots.js:: Endpoint //api/call-bot para interação com bots via OpenAI.
     + routes/auth.js:: Endpoints //api/register e //api/login para autenticação.
     + routes/clients.js:: Endpoints GET /api/clients e POST /api/clients para gestão de clientes.
     + routes/assistants.js:: Endpoints GET /api/assistants e POST /api/assistants para gestão de assistentes.
     + routes/sessions.js:: Endpoint //api/session-history/:session\_id para histórico de sessões.
     + routes/attendances.js:: Endpoints para gestão de atendimentos (ex.: POST /api/attendances).).
     + middlewares/auth.js:: Middleware authenticateToken para validar tokens JWT.
     + ..env:: Variáveis de ambiente (ex.: OPENAI\_API\_KEY, , JWT\_SECRET, , PORT).).
     + package.json:: Dependências ((express, , cors, , better-sqlite3, , jsonwebtoken, , axios, , dotenv).).
   * **Banco de Dados** ( (egide.db): ):
     + **Tabelas**:

Tabelas:

* + - * users: : user\_id, , username, , password, , role (padrão 'master'), ), master\_user\_id, , created\_at..
      * clients: : client\_id, , user\_id, , name, , cpf, , dob, , created\_at..
      * attendances: : id, , client\_id, , user\_id, , description, , created\_at, , last\_updated\_at..
      * sessions: : session\_id, , user\_id, , client\_id, , attendance\_id, , last\_updated\_at..
      * session\_history: : session\_id, , user\_id, , history (JSON com interações), created\_at..
    - **Relacionamentos**: :
      * sessions vincula user\_id a users, , client\_id a clients,, e attendance\_id a attendances(id).).
      * session\_history armazena o histórico de interações por session\_id..
  + **Endpoints Implementados**: :
    - POST /api/register:: Cria novo usuário.
    - POST /api/login:: Autentica usuário e retorna token JWT.
    - GET /api/clients:: Lista clientes do usuário logado.
    - POST /api/clients:: Cadastra novo cliente.
    - GET /api/assistants:: Lista assistentes de um Master.
    - POST /api/assistants:: Cria novo assistente.
    - POST /api/call-bot:: Chama bots (Redator, Médico, Estrategista, Supervisor) via OpenAI, registra sessão e histórico.
    - GET /api/session-history/:session\_id:: Retorna histórico de uma sessão.
    - POST /api/attendances:: Cria novo atendimento para um cliente.
  + **Funcionalidades Implementadas**: :
    - Autenticação com JWT, incluindo papéis ((master, , auxiliar).).
    - Gestão de clientes e assistentes no banco de dados.
    - Integração com OpenAI GPT-4o para geração de relatórios jurídicos.
    - Persistência de sessões e histórico em sessions e session\_history..
    - Validação de client\_id e attendance\_id em //api/call-bot..
  + **Fluxos Funcionando**: :
    - **Autenticação**: :
      * Registro de usuário via //api/register..
      * Login via //api/login,, retornando token JWT.
      * Validação de token em rotas protegidas ((authenticateToken).).
    - **Gestão de Clientes**: :
      * Cadastro de clientes com name, , cpf, , dob..
      * Listagem de clientes por user\_id..
    - **Gestão de Assistentes**: :
      * Cadastro de assistentes vinculados a um Master.
      * Listagem de assistentes por master\_user\_id..
    - **Análise Jurídica**: :
      * Requisição a //api/call-bot com role, , message, , client\_id, , attendance\_id..
      * Validação de client\_id e attendance\_id..
      * Chamada à API da OpenAI para gerar respostas.
      * Registro de sessão em sessions e histórico em session\_history..
      * Retorno da resposta ao frontend.

1. **Integração com API Externa**: :
   * **OpenAI GPT-4o**: :
     + Usada no endpoint //api/call-bot para processar mensagens e gerar relatórios.
     + Configurada via OPENAI\_API\_KEY no ..env..
     + Timeout de 90 segundos para evitar falhas ((axios).).
2. **Armazenamento**: :
   * **Banco de Dados**:: SQLite ((egide.db)) para dados estruturados (usuários, clientes, sessões, histórico).
   * **Arquivos**:: Nenhum armazenamento de arquivos implementado (upload de documentos está pendente).

**Fluxos Implementados**

1. **Fluxo de Autenticação**: :
   * Usuário acessa login.html,, insere username e password..
   * Requisição a POST /api/login retorna token JWT, salvo no localStorage..
   * Token é usado em chamadas subsequentes via createAuthHeaders ( (api.js).).
   * Logout ((auth.js)) remove o token e redireciona para login.html..
2. **Fluxo de Gestão de Clientes**: :
   * Usuário logado acessa clients.html..
   * clients.js faz GET /api/clients para listar clientes.
   * Formulário em clients.html envia POST /api/clients para cadastrar novos clientes.
   * Link "Iniciar Análise" redireciona para client-sessions.html com clientId e clientName no sessionStorage..
3. **Fluxo de Gestão de Assistentes**: :
   * Usuário Master acessa assistants.html..
   * manage-users.js faz GET /api/assistants para listar assistentes.
   * Formulário envia POST /api/assistants para criar novos assistentes.
4. **Fluxo de Análise Jurídica**: :
   * Usuário seleciona um cliente em clients.html,, redireciona para client-sessions.html..
   * Inicia nova sessão ou carrega uma existente, redirecionando para chat.html..
   * Em chat.html,, usuário insere descrição do caso.
   * mainCaseFlow ( (chat.js)) orquestra chamadas a callBotAPI ( (api.js)) para:
     + Redator: Relatório técnico.
     + Médico: Análise médica.
     + Estrategista: Análise estratégica.
     + Redator: Relatório final.
     + Supervisor: Resposta final consolidada.
   * Respostas são exibidas em ##logsIndividuais e salvas no backend ((session\_history).).
   * Sessão é registrada em sessions com client\_id e attendance\_id..
5. **Fluxo de Histórico de Sessões**: :
   * Em client-sessions.html, , client-sessions.js faz GET /api/session-history/:session\_id para listar sessões por cliente.
   * Clique em uma sessão redireciona para chat.html,, onde loadSessionData restaura o histórico.

**Arquivos Relevantes**

* **Frontend** ( (frontend/): /):
  + index.html, , chat.html, , clients.html, , client-sessions.html, , assistants.html, , login.html, , register.html..
  + js/index-page.js, , js/chat.js, , js/clients.js, , js/client-sessions.js, , js/manage-users.js, , js/api.js, , js/ui.js, , js/logs.js, , js/state.js, , js/dom-elements.js, , js/i18n.js, , js/pdf.js, , js/theme.js, , js/config.js, , js/auth.js..
  + css/styles.css..
* **Backend** ( (backend/): /):
  + server.js, , db.js..
  + routes/bots.js, , routes/auth.js, , routes/clients.js, , routes/assistants.js, , routes/sessions.js, , routes/attendances.js..
  + middlewares/auth.js..
  + ..env, , package.json..
* **Banco de Dados**: :
  + egide.db (SQLite).

**O Que Está Funcionando**

* **Autenticação**:: Login, logout, e validação de token JWT.
* **Gestão de Usuários**:: Cadastro/listagem de usuários Master e assistentes.
* **Gestão de Clientes**:: Cadastro/listagem de clientes.
* **Análise Jurídica**:: Fluxo completo com bots (Redator, Médico, Estrategista, Supervisor) gerando relatórios e respostas.
* **Persistência de Sessões**:: Salvamento e recuperação de sessões com session\_id, , client\_id, , attendance\_id..
* **Histórico de Sessões**:: Listagem e carregamento de sessões por cliente.
* **Exportação de Logs**:: Geração de PDFs a partir de logs.
* **Internacionalização**:: Suporte a português e inglês.
* **Indicador de Progresso**:: Exibição de mensagens durante chamadas à API.

**Pendências**

* **Teste Completo do Fluxo**: :
  + Validar recarregamento de chat.html para restauração da sessão.
  + Testar exclusão de logs e múltiplas sessões para o mesmo cliente.
* **Dashboard Completo**: :
  + Implementar seções detalhadas (notificações, clientes, casos).
* **Gerenciamento de Múltiplas Sessões**: :
  + Criar UI para listar/selecionar sessões.
  + Implementar GET /api/sessions..
* **Papéis/Permissões**: :
  + Adicionar controle de acesso granular (ex.: ocultar links para Auxiliares).
* **Funcionalidades Futuras**: :
  + Upload/visualização de documentos.
  + OCR para extração de texto.
  + Editor de peças jurídicas.
  + Gestão de prazos/tarefas.
  + Transcrição de áudios.

**4. Planejamento para Continuação (04/05/2025)**

4. Planejamento para Continuação (04/05/2025**)**

Para continuar amanhã, sugiro focar nas seguintes prioridades:

1. **Teste Completo do Fluxo**: :
   * Execute o fluxo "Login -> Index -> Clients -> Iniciar Análise -> Chat -> Recarregar Chat".
   * Verifique:
     + Exibição do histórico completo em ##logsIndividuais após recarregamento.
     + Registro correto de client\_id e attendance\_id em sessions..
     + Funcionalidade do botão "Limpar Logs" e criação de múltiplas sessões.
   * Compartilhe logs do frontend e backend.
2. **Melhoria do Dashboard**: :
   * Envie index.html e index-page.js para análise.
   * Proponho adicionar:
     + Lista de clientes recentes ((GET /api/clients).).
     + Lista de assistentes ((GET /api/assistants).).
     + Notificações (ex.: casos pendentes, via futuro //api/notifications).).
   * Escolha entre manter HTML/JS puro ou migrar para React com Tailwind CSS.
3. **Gerenciamento de Múltiplas Sessões**: :
   * Criar barra lateral em index.html ou clients.html para listar sessões.
   * Implementar GET /api/sessions no backend.
4. **Sincronização com GitHub**: :
   * Confirme que o repositório está atualizado com todas as mudanças de hoje.

**5. Conclusão**

Nos últimos dias (01/05/2025 a 03/05/2025), o projeto **Égide Jurídico** avançou significativamente, implementando autenticação, gestão de clientes/assistentes, vinculação cliente-sessão, e um fluxo completo de análise jurídica com IA. Hoje, resolvemos erros críticos no backend ((attendance\_id)) e no frontend (ordem de parâmetros em callBotAPI),), garantindo que o fluxo principal funcione corretamente, como comprovado pelos logs de teste.

A arquitetura atual é robusta, com frontend modular (HTML/JS, Vite, Bootstrap) e backend escalável (Node.js/Express, SQLite, OpenAI). Os fluxos de autenticação, gestão de clientes/assistentes, e análise jurídica estão operacionais, mas o teste completo e o dashboard completo são as próximas prioridades.

**Para Amanhã**::

* Realize o teste completo e compartilhe os resultados.
* Envie index.html e index-page.js para planejar melhorias no dashboard.
* Confirme a sincronização com o GitHub.

Se precisar de ajuda com o teste, ajustes no código, ou priorização de tarefas, é só me chamar no próximo chat! 😊