Manual Técnico do Sistema: Catálogo de Serviços

# 1. Visão Geral Técnica

Este documento constitui a documentação técnica oficial para a versão 1.0 do sistema de Catálogo de Serviços de TI. O seu objetivo é servir como uma referência central para a equipe de desenvolvimento, detalhando a arquitetura, a lógica de negócio, os padrões de código e os pontos de atenção para a manutenção e evolução do projeto.

# 2. Arquitetura e Padrões de Projeto

A aplicação foi desenvolvida utilizando PHP 8+ vanilla, MySQL, e JavaScript vanilla.

* **Modelo de Backend (Arquitetura Monolítica):** A aplicação segue um padrão monolítico onde a lógica de negócio, o acesso aos dados e a renderização da interface são geridos no lado do servidor. O arquivo manage/add/manage\_addservico.php é o componente central da lógica de negócio, operando como uma **Máquina de Estados Finitos (State Machine)**. O fluxo de trabalho de uma "Ficha de Serviço" é controlado por um switch que avalia o parâmetro $\_POST['acao']. Cada case representa uma transição de estado que move a ficha através do seu ciclo de vida (ex: de rascunho para em\_revisao), executando as operações de banco de dados e as validações de permissão correspondentes.
* **Modelo de Frontend (Interatividade Vanilla):** A interatividade do lado do cliente é implementada com JavaScript vanilla, com foco em melhorar a experiência do usuário sem a dependência de frameworks. As principais técnicas utilizadas são:
  1. **Manipulação Direta do DOM:** Em js/addservico.js, para a criação dinâmica de campos de formulário (diretrizes, padrões, etc.).
  2. **AJAX (fetch) para Busca Assíncrona:** Em js/script.js (ou equivalente), para a funcionalidade de busca global.
  3. **Debounce:** A lógica de busca implementa um setTimeout de 300ms para evitar requisições excessivas ao servidor durante a digitação do usuário.
* **Segurança:** A segurança é abordada em múltiplas camadas:
  1. **Validação no Cliente (JavaScript):** Fornece feedback imediato ao usuário (ex: campos obrigatórios, seleção de revisores).
  2. **Validação no Servidor (PHP):** Garante que todas as regras de negócio (ex: unicidade de e-mail, limite de caracteres) sejam cumpridas antes de qualquer operação na base de dados.
  3. **Prevenção de SQL Injection:** A aplicação utiliza Prepared Statements com bind\_param na maioria das operações de escrita e em todas as de leitura que envolvem parâmetros do usuário, o que é a prática recomendada.
  4. **Controlo de Acesso em Duas Camadas:** O sistema de permissões (detalhado abaixo) funciona tanto no frontend (desabilitando campos e escondendo botões) quanto no backend (com uma verificação final que interrompe a execução em caso de acesso indevido).

# 3. Análise Detalhada dos Componentes

## 3.1. Estrutura de Arquivos

O projeto está organizado de forma modular:

* /: Contém os arquivos da interface pública do catálogo (index.php, categoria.php, etc.).
* /manage/: Contém o painel de gerenciamento.
  + /manage/list/: Páginas de listagem de entidades.
  + /manage/add/: Formulários para criação e edição de entidades.
* /chamado/: Lógica de integração com a API do GLPI.
* /chatbot/: Lógica de integração com a API de IA (Gemini).
* conexao.php: Ponto central de conexão com a base de dados.
* buscar\_servicos.php: Endpoint para a busca AJAX.

## 3.2. Lógica de Backend (PHP)

* **conexao.php:**
  + **Função:** Estabelece a conexão com o banco de dados via mysqli.
  + **Ponto de Atenção:** As credenciais estão "hardcoded". Para produção, estas devem ser movidas para variáveis de ambiente (getenv()) para evitar a sua exposição no repositório de código.
* **manage/add/manage\_addservico.php:**
  + **Lógica de Permissão:** A permissão de edição é controlada pela variável booleana $pode\_editar. O seu valor é definido no início do script, com base numa série de regras que cruzam o tipo\_usuario, o status\_atual da ficha e o current\_user\_id (simulado via $\_GET).
  + **Segurança de Acesso:** A barreira de segurança no POST (dentro dos cases aprovar\_revisor e aprovar\_po) é crucial. Ela revalida a permissão do usuário contra os dados atuais do banco antes de executar a ação, prevenindo acessos forçados.
* **buscar\_servicos.php:**
  + **Função:** Endpoint que recebe um termo de busca via $\_GET.
  + **Lógica:** Constrói uma cláusula WHERE dinâmica com múltiplas condições LIKE para buscar o termo em vários campos (Titulo, Descricao, etc.).
  + **Segurança:** A query principal utiliza prepared statements, o que é a prática correta.
* **Integrações (/chamado e /chatbot):**
  + **Padrão de Projeto (Proxy):** Estes scripts funcionam como um proxy. O frontend envia a requisição para eles, e eles adicionam as credenciais/chaves de API (que estão seguras no servidor) e repassam a requisição para os serviços externos (GLPI e Gemini). Isto evita a exposição de chaves de API no código do cliente.

## 3.3. Lógica de Frontend (JavaScript)

* **js/addservico.js:**
  + **Validação por Clique:** A validação de campos foi corretamente movida de um listener de submit genérico para listeners de click específicos nos botões. Isto resolveu os conflitos de eventos e garantiu que todos os botões funcionem de forma independente.
  + **Justificativa com prompt():** A funcionalidade de justificativa utiliza o prompt() nativo do navegador. Esta abordagem é a mais simples e robusta, pois elimina a necessidade de um segundo formulário ou de um div escondido, que era a fonte dos bugs.

## 3.4. Banco de Dados

A estrutura é relacional, com as seguintes tabelas principais:

* servico: Armazena a ficha principal do serviço, incluindo o seu status.
* categoria, subcategoria: Estrutura hierárquica.
* diretriz, itemdiretriz: Relação um-para-muitos.
* padrao, itempadrao: Relação um-para-muitos.
* checklist: Itens de verificação para um serviço.
* pos, revisores: Tabelas de usuários.
* servico\_revisores: Tabela de junção (muitos-para-muitos) que associa quais revisores foram designados para qual serviço.
* sugestoes: Nova tabela com chave estrangeira servico\_id e a regra ON DELETE CASCADE para garantir a integridade referencial.

# 4. Roteiro de Implementação e Próximos Passos

Esta seção descreve o escopo e as diretrizes para a integração final do projeto ao ambiente do GLPI.

## 4.1. Banco de Dados e Conexão:

* + **Estrutura de Dados:** Implementar as tabelas customizadas no banco de dados do GLPI.
  + **Conexão:** Configurar o conexao.php para apontar para o banco de dados do ambiente correspondente.

## 4.2. Sistema de Autenticação:

* + **Integração com GLPI:** Substituir a autenticação local atual ($\_SESSION['username']) por uma que valide os usuários e seus perfis diretamente no GLPI.
  + **Perfis de Acesso:** O sistema deverá reconhecer os perfis ADMIN e PO (já existentes) e um novo perfil "Revisor", que precisa de ser criado no GLPI.
  + **Refatoração:** Com a nova autenticação, os recursos para listar e adicionar POs e Revisores na aplicação serão descontinuados.

## 4.3. Aplicação e Workflow:

* + **Painel de Debug:** O painel de debug deve ser ocultado ou removido na versão de produção.
  + **Workflow das Fichas:** O workflow implementado em manage\_addservico.php deve ser validado com os pontos focais do negócio.
  + **Criação de Chamados via API:** Configurar os scripts (glpi\_api.php, processar\_chamado.php) para utilizar a API do GLPI em produção.

## 4.4. Implantação e Integração Final:

* + **Hospedagem:** A aplicação será hospedada num ambiente externo ao GLPI.
  + **Acesso via GLPI:** A integração será feita através de um plugin no GLPI que inserirá um botão de redirecionamento para a aplicação do catálogo.

## 4.5. Inteligência Artificial (Chatbot):

* + **Modelo de IA:** Avaliar a substituição da API do Gemini pelo modelo PHI-2, considerando os custos e a infraestrutura necessária.
  + **Análise de Viabilidade:** A viabilidade da implementação do PHI-2 deve ser discutida com o responsável pela infraestrutura.