UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PABLO ANDREI PEREIRA

SERVICEBOOK 3.0: EVOLUÇÃO DE ARQUITETURA MONOLÍTICA PARA API REST E SINGLE PAGE APPLICATION (SPA)

GUARAPUAVA

2026

PABLO ANDREI PEREIRA

SERVICEBOOK 3.0: EVOLUÇÃO DE ARQUITETURA MONOLÍTICA PARA API REST E SINGLE PAGE APPLICATION (SPA)

Servicebook 3.0: Evolution from Monolithic Architecture to REST API and Single Page Application (SPA)

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Roni Fabio Banaszewski

Coorientador: Tais Michele Hryssai da Luz

GUARAPUAVA

2026

RESUMO

O projeto acadêmico ServiceBook, uma aplicação web de intermediação de serviços locais, opera sob uma arquitetura monolítica legada, *JavaServer Pages* (JSP) e Spring *Model-View-Controller* (Modelo-Visão-Controlador) (MVC), que limita sua escalabilidade e evolução visual. Para solucionar essas restrições, este trabalho propõe a modernização do sistema para um modelo totalmente desacoplado, por meio da construção de uma API *Representational State Transfer* (Transferência de Estado Representacional) (REST) com Spring Boot e uma Single Page Application (SPA) reativa com Angular. A reengenharia será guiada por práticas ágeis e inovadoras, com destaque para o Desenvolvimento Agêntico (auxiliado por Inteligência Artificial), *Specification-Driven Development* (Desenvolvimento Orientado a Especificações) (SDD) e *Test-Driven Development* (Desenvolvimento Orientado a Testes) (TDD). Além de sanar os gargalos de acoplamento da versão atual, o projeto prevê a adoção de um novo *Design System* inspirado em grandes *marketplaces* para "produtizar" a exibição dos serviços, bem como a integração de recursos de Inteligência Artificial (IA) para a moderação e sumarização de avaliações. Como resultado esperado, almeja-se que a nova versão do ServiceBook entregue uma experiência de usuário (UX) mais fluida, moderna e que facilite futuras expansões.

Palavras-chave: modernização de sistemas; single page application; desenvolvimento agêntico; arquitetura desacoplada; contratação de serviços.

ABSTRACT

The ServiceBook academic project, a local service intermediation platform, operates under a legacy monolithic architecture (JSP and Spring MVC) that limits its scalability and visual evolution. To address these constraints, this work proposes the modernization of the system into a fully decoupled model through the construction of a RESTful API using Spring Boot and a reactive Single Page Application (SPA) with Angular. The reengineering will be guided by agile and innovative practices, highlighting Agentic Development (AI-assisted), Spec-Driven Development (SDD), and Test-Driven Development (TDD). In addition to resolving the coupling bottlenecks of the current version, the project foresees the adoption of a new Design System inspired by major marketplaces to ”productize” service displays, as well as the integration of AI features for content moderation and review summarization. As an expected result, the goal is for the new version of ServiceBook to deliver a more fluid and modern user experience (UX), facilitating future expansions.

Keywords: system modernization; single page application; agentic development; decoupled architecture; service contracting.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ACID	*Atomicity, Consistency, Isolation, Durability* (Atomicidade, Consistência, Isolamento e Durabilidade)
API	*Application Programming Interface* (Interface de Programação de Aplicações)
BDD	*Behavior-Driven Development* (Desenvolvimento Orientado a Comportamento)
CD	*Continuous Deployment* / *Continuous Delivery* (Implantação Contínua / Entrega Contínua)
CI	*Continuous Integration* (Integração Contínua)
CSS	*Cascading Style Sheets* (Folhas de Estilo em Cascata)
DOM	*Document Object Model* (Modelo de Objeto de Documentos)
DTO	*Data Transfer Object* (Objeto de Transferência de Dados)
HTML	*HyperText Markup Language* (Linguagem de Marcação de Hipertexto)
IA	Inteligência Artificial
ID	Identificador
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
IPEA	Instituto de Pesquisa Econômica Aplicada
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation* (Notação de Objetos JavaScript)
JSP	*JavaServer Pages*
JWT	*JSON Web Token*
LLM	*Large Language Model* (Grande Modelo de Linguagem)
MCP	*Model Context Protocol* (Protocolo de Contexto de Modelo)
MoSCoW	*Must have, Should have, Could have, Won't have* (Tem que ter, Deveria ter, Poderia ter, Não vai ter)

MVC	*Model-View-Controller* (Modelo-Visão-Controlador)
PIB	Produto Interno Bruto
PR	*Pull Request* (Pedido de Mesclagem)
PRD	*Product Requirements Document* (Documento de Requisitos do Produto)
PWA	*Progressive Web App* (Aplicação Web Progressiva)
RAM	*Random Access Memory)* (Memória de Acesso Aleatório.)
REST	*Representational State Transfer* (Transferência de Estado Representacional)
SaaS	*Software as a Service* (Software como Serviço)
SDD	*Specification-Driven Development* (Desenvolvimento Orientado a Especificações)
SPA	*Single Page Application* (Aplicação de Página Única)
SSR	*Server-Side Rendering* (Renderização do Lado do Servidor)
TCC	Trabalho de Conclusão de Curso
TDD	*Test-Driven Development* (Desenvolvimento Orientado a Testes)
TSI	Tecnologia em Sistemas para Internet
UI	*User Interface* (Interface de Usuário)
UTFPR	Universidade Tecnológica Federal do Paraná
UX	*User Experience* (Experiência do Usuário)

SUMÁRIO

1 INTRODUÇÃO

O setor de serviços desempenha um papel fundamental e consolidado como a espinha dorsal da economia brasileira. Representando cerca de 70% do Produto Interno Bruto (PIB) nacional, o setor registrou um crescimento de 3,7% em 2024 — o maior avanço desde 2021. Esse vigor econômico se manteve ao longo do tempo, com a atividade de serviços atingindo níveis historicamente elevados, como o crescimento de 0,5% observado em fevereiro de 2026 em relação ao mesmo período do ano anterior (ESTATíSTICA, 2025).

Apesar de sua magnitude e relevância, a prestação de serviços no Brasil enfrenta desafios estruturais significativos. Cerca de 40% da força de trabalho brasileira atua na informalidade (Fundação Getúlio Vargas, 2025). Essa característica gera um ambiente fragmentado, no qual clientes encontram dificuldades para localizar profissionais qualificados e confiáveis, enquanto os prestadores de serviço carecem de ferramentas adequadas para divulgar seu trabalho, gerenciar agendas e precificar suas atividades.

É exatamente para preencher essa lacuna de confiança e organização que as plataformas digitais ganharam protagonismo. Segundo o Instituto de Pesquisa Econômica Aplicada (IPEA), o Brasil conta com aproximadamente 1,4 milhão de trabalhadores inseridos na chamada *gig economy* (trabalhos temporários ou sob demanda), um número 60% superior ao registrado em 2016 (Instituto de Pesquisa Econômica Aplicada, 2025). Esses trabalhadores e clientes dependem cada vez mais destes sistemas que ofereçam mecanismos de contratação, avaliação e rastreabilidade.

Visando tratar essa problemática, o projeto ServiceBook foi idealizado pelo professor orientador deste trabalho. Inicialmente, o sistema começou a ser desenvolvido por alunos da disciplina de Desenvolvimento para Web 4, vinculada ao Curso Superior de Tecnologia em Sistemas para Internet (TSI) da Universidade Tecnológica Federal do Paraná (UTFPR), campus Guarapuava. Mais adiante, a iniciativa ganhou continuidade e evoluiu no Trabalho de Conclusão de Curso (TCC) da aluna Taís Luz (LUZ, 2024). Em seu estado atual, o ServiceBook tem como objetivo central conectar clientes a profissionais e empresas prestadoras de serviços, suportando dois cenários de contratação: a busca passiva, em que o cliente publica uma demanda e aguarda candidaturas de profissionais; e a busca ativa, em que o cliente localiza o prestador desejado, consulta a agenda e tabela de preços, se houver, e inicia o contato via plataforma.

Ao longo do tempo, o ServiceBook incorporou diversas funcionalidades importantes. Contudo, por ser um projeto de natureza acadêmica, seu desenvolvimento inicial ocorreu de forma fragmentada, sendo construído a “múltiplas mãos” por turmas de alunos que ainda estavam em pleno processo de formação técnica. Naturalmente, por não se tratar de uma equipe de profissionais já consolidados no mercado, as sucessivas trocas de desenvolvedores fizeram com que o sistema acumulasse problemas estruturais e inconsistências visuais.

Nesse contexto, a etapa desenvolvida no TCC de (LUZ, 2024) representou um salto significativo de qualidade. Ao assumir o projeto de forma individual, a aluna pôde centralizar o

desenvolvimento, trabalhando com maior foco e livre das interferências paralelas e sobreposições de código que eram comuns durante as aulas da disciplina. Isso trouxe uma coesão muito maior para a plataforma.

Apesar do importante avanço organizacional, a base tecnológica herdada manteve o sistema restrito a uma arquitetura monolítica clássica, caracterizada pelo forte acoplamento entre a camada de apresentação e o servidor. Essa dependência estrutural eleva a complexidade de manutenção, prejudica a cobertura de testes e restringe a escalabilidade da plataforma, uma vez que o servidor assume a sobrecarga de processar as regras de negócio e renderizar a estrutura gráfica *HyperText Markup Language* (Linguagem de Marcação de Hipertexto) (HTML) (Sociedade Brasileira de Computação, 2024; LOPES, 2021).

Sob a ótica de usabilidade, o desenvolvimento fragmentado ao longo do tempo resultou em uma interface visualmente inconsistente. Embora o framework adotado originalmente, o Materialize, seja uma ferramenta robusta e eficiente, ele impõe uma identidade visual rígida. Essa limitação de customização restringe a evolução estética da plataforma, tornando evidente a necessidade de transição para uma abordagem que ofereça a flexibilidade necessária para a consolidação de um Design System próprio, independente e alinhado aos grandes marketplaces contemporâneos.

Para solucionar essas restrições, torna-se evidente a necessidade de uma profunda refatoração estrutural no sistema. O caminho técnico mais adequado consiste em desestruturar a arquitetura monolítica atual, convertendo a camada do servidor em uma *Application Programming Interface* (Interface de Programação de Aplicações) (API) REST independente. No lado do *front-end*, a adoção de uma arquitetura baseada em componentes apresenta-se como a solução ideal para mitigar a despadronização visual, viabilizando o reúso de código e assegurando que todas as telas sigam diretrizes consistentes.

Além disso, o desacoplamento entre a interface e o servidor proporciona a flexibilidade necessária para repensar o *design* da plataforma. Essa separação facilita a adoção de tecnologias de estilização modernas, permitindo a criação de uma experiência fluida — com uma navegação inspirada em grandes *e-commerces*, mas perfeitamente adaptada à dinâmica de contratação de serviços. Por fim, essa nova base técnica prepara o ecossistema para a incorporação de recursos avançados, como a integração de modelos de Inteligência Artificial.

Sendo assim, este trabalho propõe analisar o cenário atual do *ServiceBook* e aplicar essa modernização na sua arquitetura e interface, buscando transformar o projeto em uma plataforma padronizada e fácil de manter.

1.1 Objetivos

Nesta seção serão apresentados o objetivo geral e os objetivos específicos do presente trabalho.

1.1.1 Objetivo geral

Modernizar a aplicação web ServiceBook por meio da migração de sua arquitetura monolítica para um modelo desacoplado, baseado em API REST e *Single Page Application* (Aplicação de Página Única) (SPA), visando aprimorar a escalabilidade, a manutenibilidade e a experiência do usuário.

1.1.2 Objetivos específicos

- Analisar a arquitetura atual do ServiceBook, identificando os acoplamentos críticos entre a camada de apresentação e a camada de negócios, bem como os principais gargalos de manutenção e escalabilidade.
- Reestruturar o back-end como uma API REST, promovendo maior desacoplamento, reutilização e escalabilidade do sistema.
- Recriar o front-end como uma SPA orientada a componentes, consumindo exclusivamente a API REST e eliminando a necessidade de renderização no servidor.
- Desenvolver um Design System, estabelecendo padrões visuais, componentes reutilizáveis e melhorias na experiência do usuário.
- Integrar um *Large Language Model* (Grande Modelo de Linguagem) (LLM) para sumarizar avaliações de profissionais, com o objetivo de facilitar a interpretação das informações e apoiar a tomada de decisão dos usuários.

2 TRABALHOS RELACIONADOS

Neste capítulo, é realizada uma análise comparativa (*benchmarking*) de plataformas consolidadas no mercado digital. O objetivo consiste em identificar padrões de interface de usuário *User Interface* (Interface de Usuário) (UI), experiência do usuário *User Experience* (Experiência do Usuário) (UX), fluxos de navegação e recursos tecnológicos que possam servir de referência para o processo de modernização do ServiceBook.

A análise abrange desde grandes plataformas de comércio eletrônico quanto sistemas específicos de intermediação de serviços, buscando compreender soluções adotadas em termos de organização visual, usabilidade, navegação e interação entre usuários.

2.1 Plataformas de Comércio Eletrônico de Produtos

Embora o ServiceBook seja voltado à intermediação de serviços, a adoção de padrões visuais utilizados em plataformas consolidadas de comércio eletrônico representa uma estratégia relevante para transmitir confiança, melhorar a experiência do usuário e facilitar os mecanismos de busca ativa dentro da aplicação.

A crescente tendência de “produtização” de serviços faz com que profissionais e empresas passem a ser apresentados de maneira semelhante a produtos em vitrines digitais. Nesse contexto, elementos como organização visual, hierarquia de informações, avaliações, categorização e mecanismos de recomendação tornam-se referências importantes para a modernização da interface do ServiceBook.

2.1.1 Shopee: Padronização Visual e Descoberta

A Shopee destaca-se pela capacidade de organizar uma grande quantidade de informações e produtos em uma única interface sem comprometer a navegação do usuário. Dessa forma, a plataforma torna-se uma referência relevante no processo de modernização visual do ServiceBook.

- **Pontos Positivos:** O uso de cartões (*cards*) organizados em grade (*grid*) permite que múltiplas opções sejam visualizadas simultaneamente, reduzindo a necessidade de rolagem excessiva. Além disso, a separação da interface inicial em categorias e blocos temáticos, acompanhados de ações como “Ver Mais”, favorece a navegação exploratória e facilita a descoberta de conteúdos, especialmente para usuários visitantes que ainda não possuem um objetivo específico definido.
- **Pontos Negativos:** A elevada quantidade de estímulos visuais, banners promocionais e elementos simultâneos pode resultar em uma interface visualmente poluída. Em plataformas voltadas à contratação de serviços, como o ServiceBook, a utilização exces-

siva desses recursos pode comprometer a clareza das informações e a percepção de confiabilidade durante o processo de contratação.

Conforme ilustrado na Figura 1, a plataforma organiza os produtos por meio de cartões distribuídos em grade e categorias temáticas, facilitando a navegação exploratória do usuário.

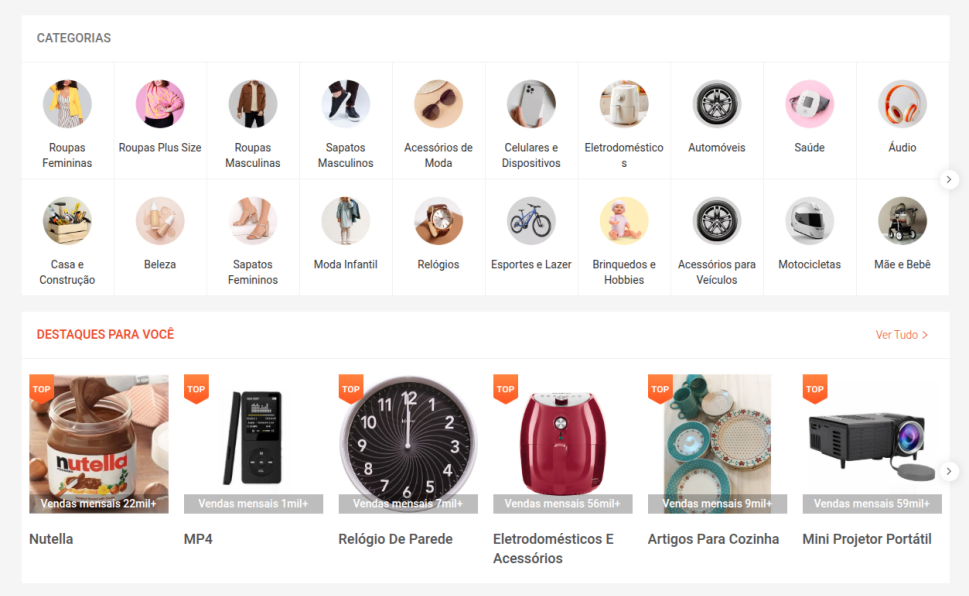

Figura 1 – Interface da Shopee organizada em grade de cartões e categorias.
Fonte: (Shopee, 2026).

2.1.2 Mercado Livre: Inteligência Artificial e Credibilidade

O Mercado Livre destaca-se como uma das principais referências nacionais em mecanismos de confiança entre compradores e vendedores em plataformas digitais. Essa característica torna-se especialmente relevante para o contexto do ServiceBook, considerando que a contratação de serviços envolve elevado grau de subjetividade e dependência da reputação dos profissionais.

- **Pontos Positivos:** O recurso de sumarização de avaliações por Inteligência Artificial representa um diferencial tecnológico relevante. Em vez de exigir que o usuário leia uma grande quantidade de comentários individualmente, a plataforma utiliza modelos de linguagem para consolidar os principais aspectos positivos e negativos em um painel resumido. Além disso, a dinâmica de pagamento antecipado, embora seja o padrão consolidado no comércio de produtos, atua como uma referência importante para romper o paradigma tradicional do setor de serviços (onde o acerto costuma ocorrer apenas após a execução). Esse fluxo de retenção de valor inspira o modelo adotado pelo ServiceBook para a contratação prévia de serviços com preços tabelados, garantindo o compromisso entre as partes e a reserva segura da agenda do profissional.

- **Pontos Negativos:** Embora a plataforma organize seu vasto catálogo através de um amplo sistema de categorias, a navegação pode tornar-se complexa para usuários iniciantes ao atingir níveis hierárquicos muito profundos. Devido a essa extensa ramificação de menus, a barra de pesquisa acaba assumindo um papel fundamental e quase obrigatório para facilitar a localização direta de itens, reduzindo a dependência da exploração manual pelos usuários.

Conforme ilustrado na Figura 2, a plataforma utiliza recursos de Inteligência Artificial para sintetizar avaliações realizadas pelos usuários, apresentando resumos com os principais pontos positivos e negativos identificados nos comentários.

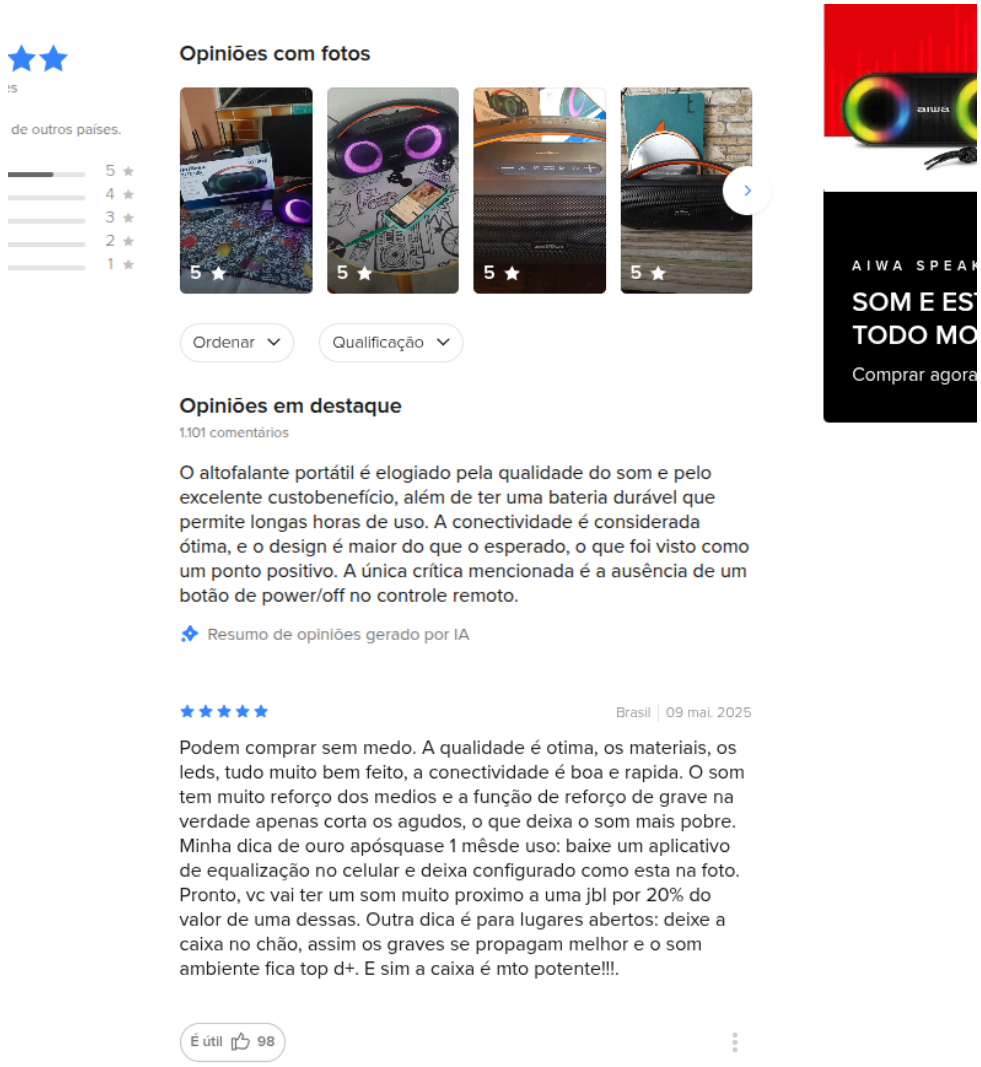

Figura 2 – Sumarização de avaliações de usuários gerada por Inteligência Artificial no Mercado Livre.

Fonte: (Mercado Livre, 2026).

2.2 Plataformas de Intermediação de Serviços

A análise de plataformas pertencentes ao mesmo segmento de mercado permite compreender como os sistemas atuais realizam a intermediação entre clientes e prestadores de serviço. Além disso, o estudo dessas soluções possibilita identificar padrões de navegação, fluxos de contratação, mecanismos de confiança e estratégias de experiência do usuário aplicadas em plataformas já consolidadas no setor.

2.2.1 GetNinjas: O Modelo de Busca Nacional

O GetNinjas consolidou-se como uma das maiores plataformas de intermediação de serviços do Brasil, atuando em larga escala nacional. O sistema opera conectando clientes que necessitam de demandas variadas — desde manutenções residenciais até consultorias e aulas particulares — a profissionais autônomos e empresas locais. A análise desta plataforma torna-se fundamental para este trabalho por representar o principal ecossistema de geração de distribuição de pedidos de orçamento e busca passiva do mercado brasileiro, oferecendo um excelente parâmetro de comparação para os fluxos de contratação que o *ServiceBook* propõe modernizar.

- **Pontos Positivos:** O fluxo de cadastro de solicitações é altamente otimizado por meio de formulários guiados em etapas (*wizards*). A plataforma realiza perguntas contextuais e específicas de acordo com o serviço selecionado, reduzindo a carga cognitiva do usuário e evitando descrições excessivamente manuais. Essa abordagem melhora a precisão na coleta das necessidades do cliente e contribui para o encaminhamento mais adequado das solicitações aos profissionais disponíveis na região.
- **Pontos Negativos:** A plataforma apresenta maior foco no modelo de busca passiva. Em grande parte do fluxo, o cliente não possui liberdade para selecionar previamente um profissional por meio de uma vitrine visual antes da abertura da solicitação. Esse comportamento limita a autonomia de usuários que desejam realizar buscas ativas e comparar profissionais diretamente pela interface.

Conforme ilustrado na Figura 3, a interface inicial da plataforma prioriza mecanismos de busca rápida e categorização de serviços, direcionando o usuário para um fluxo guiado de solicitação.

2.2.2 Pega Empreita: Referência Regional

O Pega Empreita configura-se como uma plataforma regional relevante no contexto de Guarapuava-PR, operando sob o modelo estrutural de **busca ativa**. Seu foco principal é a in-

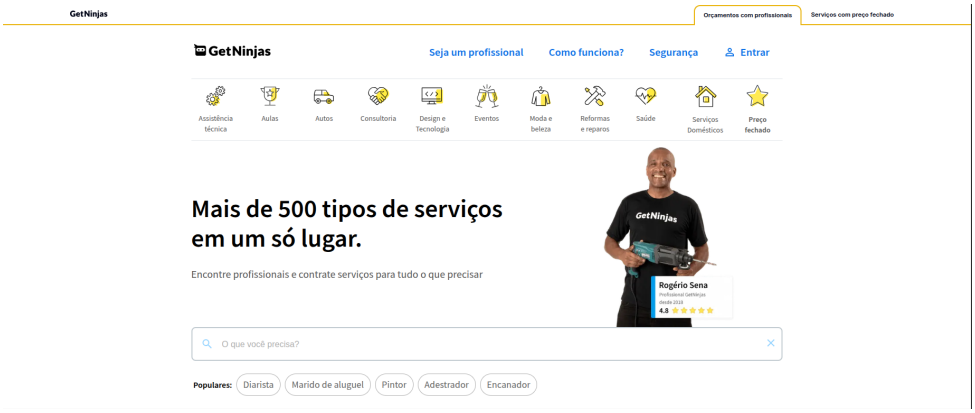

Figura 3 – Interface inicial da plataforma GetNinjas com destaque para busca e categorização de serviços.

Fonte: (GetNinjas, 2026).

termediação de profissionais voltados aos setores de construção civil, manutenção e reparos domésticos. A análise desta plataforma torna-se pertinente para este trabalho por representar uma solução inserida no mesmo contexto regional do *ServiceBook*, permitindo observar estratégias locais de navegação, apresentação de serviços e a dinâmica de contato direto entre clientes e prestadores.

- **Pontos Positivos:** Ao focar na busca ativa, o sistema concede autonomia ao cliente, que pesquisa exatamente o que necessita e visualiza de imediato a listagem de opções disponíveis. A plataforma apresenta uma segmentação de mercado bem definida, utiliza uma linguagem alinhada ao contexto regional e disponibiliza filtros específicos para o nicho de atuação, facilitando a tomada de decisão rápida do usuário.

- **Pontos Negativos:** A plataforma atua estritamente como um portal de classificados digitais, monetizado por meio de planos de assinatura (mensais a anuais) cobrados dos profissionais para a exibição de seus anúncios. Devido a essa natureza arquitetural, o sistema apresenta lacunas funcionais significativas. Não há suporte a portfólios estruturados nem intermediação de pagamentos intraplataforma, visto que a negociação é totalmente descentralizada e redirecionada para o WhatsApp. Ademais, o mecanismo de reputação apresenta uma falha crítica de confiabilidade: qualquer usuário autenticado (via conta Google, por exemplo) pode registrar avaliações no perfil de um trabalhador, sem a exigência de vínculo ou comprovação de que o serviço foi efetivamente contratado e prestado. Essa ausência de controle sobre o ciclo de vida da solicitação fragiliza a segurança do contratante.

Conforme ilustrado na Figura 4, a interface inicial da plataforma prioriza mecanismos de busca rápida, categorização de serviços específicos do nicho de atuação e uma seção destinada ao destaque de profissionais da região. Essa abordagem direciona o usuário para uma navegação exploratória objetiva, cujo propósito final é fornecer o atalho para a comunicação direta e imediata com o profissional via WhatsApp.

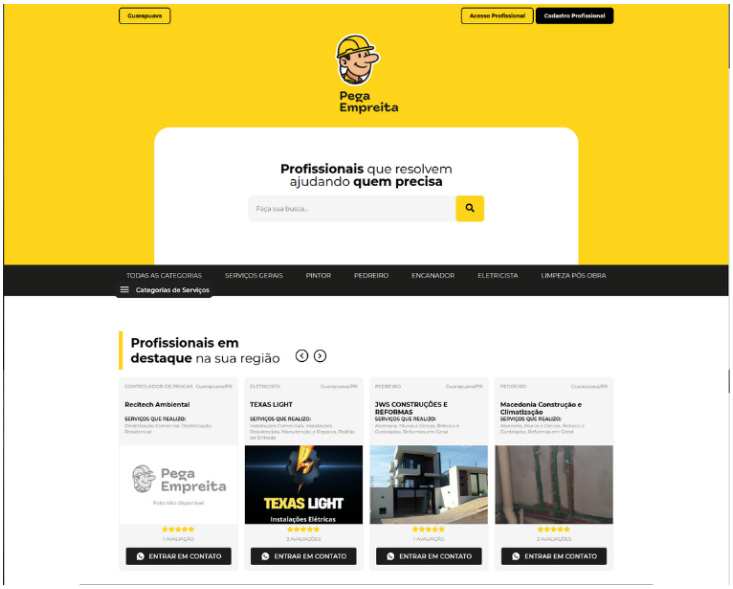

Figura 4 – Interface inicial da plataforma Pega Empreita com destaque para busca de serviços e profissionais regionais.

Fonte: (Pega Empreita, 2026).

3 CONTEXTUALIZAÇÃO E DIAGNÓSTICO DO SERVICEBOOK

Para compreender a necessidade de modernização proposta neste trabalho, é fundamental analisar a evolução do ServiceBook, desde a sua concepção até o seu estado atual. Este capítulo apresenta o histórico de desenvolvimento da plataforma, descreve a arquitetura e as tecnologias utilizadas nas versões anteriores e, por fim, realiza um diagnóstico técnico e visual das principais limitações identificadas, justificando a proposta de refatoração para uma arquiteura desacoplada e baseada em componentes.

3.1 Histórico e Evolução Funcional

O ServiceBook surgiu como um projeto acadêmico idealizado pelo Prof. Roni Fábio Banaszewski, orientador deste trabalho, e foi desenvolvido inicialmente no âmbito da disciplina de Desenvolvimento para Web 4, do Curso Superior de Tecnologia em Sistemas para Internet da UTFPR, campus Guarapuava. As primeiras versões do sistema foram construídas de forma colaborativa por diferentes equipes de alunos ao longo das edições da disciplina.

A versão inicial estabeleceu a base estrutural da plataforma, sendo voltada exclusivamente ao modelo de contratação por **busca passiva**. Nesse modelo, o fluxo de utilização do sistema contemplava as seguintes funcionalidades:

- **Gestão de Usuários:** Cadastro segmentado entre clientes e profissionais autônomos, incluindo validação das informações de contato.
- **Publicação de Demandas:** Capacidade de o cliente criar anúncios detalhados contendo as especialidades requeridas, a descrição do problema, o prazo desejado e o anexo de mídias para auxiliar na compreensão do serviço solicitado.
- **Mecanismo de Candidatura:** Permitia que profissionais visualizassem anúncios compatíveis com suas competências em um painel e registrassem interesse nos serviços desejados.
- **Gestão de Status:** O cliente analisava as candidaturas recebidas, selecionava o profissional desejado e alterava o status da solicitação conforme o andamento do serviço.

Posteriormente, o projeto passou por uma evolução significativa durante o Trabalho de Conclusão de Curso desenvolvido por (LUZ, 2024). Ao assumir o desenvolvimento da plataforma de forma individual, a autora expandiu as regras de negócio existentes e incorporou novas funcionalidades voltadas ao processo de contratação de serviços. Entre as principais contribuições introduzidas nessa versão, destacam-se:

- **Busca Ativa:** Possibilitou que os clientes buscassem diretamente por profissionais através de uma hierarquia composta por Categorias, Especialidades e Serviços específicos, sem a obrigatoriedade de criação prévia de um anúncio.
- **Perfil de Empresas:** Introduziu a possibilidade de cadastro de empresas prestadoras de serviço e o gerenciamento de seus respectivos quadros de funcionários.
- **Módulo de Avaliações e Portfólio:** Implementou um sistema de avaliação no qual clientes podem avaliar os serviços prestados por meio de notas, comentários de texto e mídias (fotos), gerando um portfólio público para auxiliar futuros contratantes.
- **Agendamento e Pagamento:** Adicionou fluxos para visualização de tabelas de preços e uma simulação de módulo de pagamentos.

Apesar da inegável evolução funcional e da consolidação das regras de negócio promovidas por (LUZ, 2024), a base tecnológica da plataforma permaneceu vinculada as decisões arquiteturais definidas nas versões iniciais do projeto.

3.2 Arquitetura e Tecnologias Legadas

O estado atual do ServiceBook utiliza uma arquitetura monolítica clássica. O sistema utiliza o ecossistema Spring (Spring Framework, Spring Boot, Spring Data e Spring Security) e foi desenvolvido na linguagem Java.

Na camada de apresentação, a aplicação adota o padrão *Server-Side Rendering* (Renderização do Lado do Servidor) (SSR), no qual as interfaces visuais são processadas diretamente no servidor. As páginas são geradas dinamicamente por meio de tecnologias de *template engine*, como JSP (*JavaServer Pages*) e, em alguns módulos, Thymeleaf, responsáveis por integrar código HTML aos dados processados pelos controladores da aplicação desenvolvidos em Java.

Para adicionar dinamismo às páginas no navegador do cliente, o sistema utiliza *JavaScript* puro e a biblioteca jQuery, responsáveis pela manipulação do *Document Object Model* (Modelo de Objeto de Documentos) (DOM) e pela realização de requisições assíncronas parciais. A estilização visual da aplicação é baseada no *framework* Materialize, responsável por implementar componentes seguindo as diretrizes do *Material Design*.

3.3 Diagnóstico do Sistema: Limitações e Desafios

Embora a arquitetura atual tenha sido suficiente para validar o conceito da plataforma e atender as demandas acadêmicas anteriores, uma análise técnica evidencia limitações estruturais e visuais que dificultam a escalabilidade do sistema e impactam negativamente na experiência do usuário.

3.3.1 Alto Acoplamento, Dificuldade de Manutenção e Gargalos de Escalabilidade

A principal limitação arquitetural do ServiceBook reside no forte acoplamento entre a camada de visão JSP e a camada de controle (Spring MVC). Como as páginas são processadas no servidor, qualquer alteração simples na interface visual exige a recompilação e o reimplante de toda a aplicação back-end.

Essa dependência afeta diretamente a escalabilidade do sistema. No modelo de SSR, o back-end é responsável por uma "jornada dupla": além de executar cálculos de regras de negócio, processar validações e realizar consultas ao banco de dados, o servidor também precisa alocar recursos computacionais para montar toda a estrutura gráfica HTML antes de enviá-la ao cliente. Em um cenário de aumento de tráfego, esse processamento visual concorrente acelera a sobrecarga do servidor, limitando a capacidade da plataforma de escalar para um grande volume de acessos simultâneos.

Somado a isso, há o gargalo do gerenciamento de sessões. Aplicações web monolíticas tradicionais operam sob o modelo *stateful*, armazenando os dados de sessão e autenticação diretamente na memória *Random Access Memory)* (Memória de Acesso Aleatório.) (RAM) do servidor. À medida que o número de usuários simultâneos cresce, o consumo de memória aumenta exponencialmente, dificultando o escalonamento horizontal (adição de novos servidores) devido à necessidade de sincronizar essas sessões entre diferentes instâncias. A transição para uma API REST, que utiliza comunicação *stateless* (sem manutenção de estado no servidor) e autenticação baseada em *tokens* (como *JSON Web Token* (JWT)), elimina esse gargalo, permitindo que as requisições sejam distribuídas livremente por balanceadores de carga sem perda de contexto.

Além disso, o uso de jQuery distribuído por diversos arquivos para manipulação do estado das telas resulta em um código *front-end* imperativo e de difícil rastreabilidade. Em fluxos que exigem maior interatividade — como o cadastro de anúncios em múltiplas etapas e a transição entre os estados de contratação (*Disponível*, *Para Orçamento* e *Em Execução*) —, a arquitetura legada depende de manipulações manuais do DOM via jQuery.

A ausência de um gerenciamento de estado centralizado e reativo dificulta a sincronização entre as regras de negócio e a interface, aumentando a complexidade de manutenção e o risco de inconsistências visuais durante a navegação do usuário.

A complexidade das regras de apresentação da interface legada torna-se evidente no gerenciamento de estados de uma solicitação de um serviço. Conforme ilustrado na Figura 5, uma única ação pode desencadear alterações em diferentes painéis da aplicação. Quando o cliente publica um anúncio de uma demanda, ele consta em sua aba de “Disponíveis”. Ao ocorrer uma candidatura, o anúncio passa a ser exibida como “Em Disputa” para o profissional. A partir do momento em que o cliente seleciona o candidato, o fluxo transita para “Para Orçamento” na visão do contratante e “Para Confirmar” na visão do prestador, exigindo uma sincronização contínua até a conclusão (“Concluídos” e “Executados”, respectivamente).

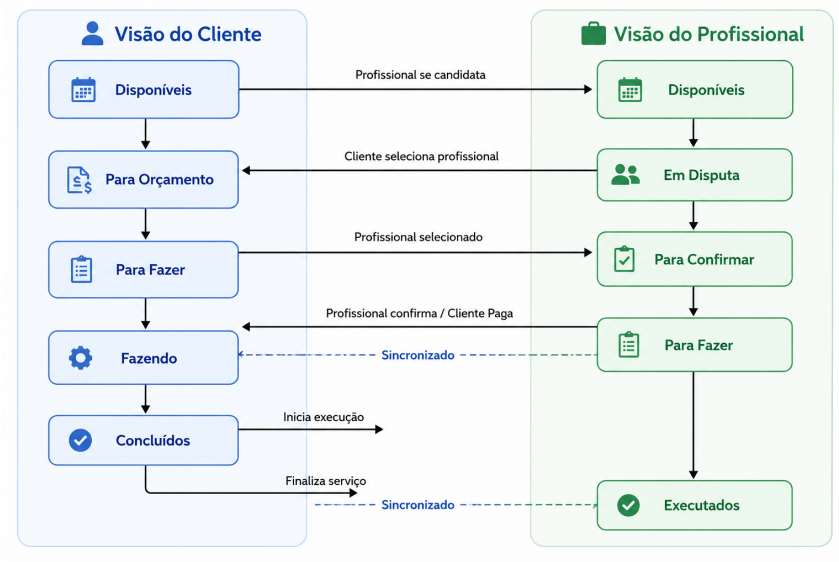

Figura 5 – Mudanças de estados de um serviço

Fonte: Autor (2026).

No modelo arquitetural legado, baseado em SSR com páginas JSP e manipulações via jQuery, manter a sincronização entre os estados das abas exigia constantes recarregamentos de página ou implementações complexas no *front-end*. Essa dificuldade em garantir a consistência visual em tempo real evidencia as limitações da arquitetura atual e reforça a necessidade de adoção de *frameworks* reativos modernos, capazes de gerenciar o estado global da aplicação de forma centralizada e dinâmica refletindo imediatamente as alterações de status na interface dos diferentes usuários.

Adicionalmente, a estrutura do código legado apresenta limitações significativas para o ciclo de desenvolvimento e expansão. A mistura física de artefatos de back-end e front-end no mesmo projeto, somada ao uso de jQuery para manipulação manual do DOM, dificulta a governança do código e torna a automação de testes unitários um desafio complexo. Como a lógica de apresentação e as regras de negócio encontram-se frequentemente entrelaçadas, a criação de testes isolados torna-se inviável, forçando o uso de testes de integração lentos e frágeis.

Por fim, o modelo fundamentado na geração de HTML pelo servidor limita a plataforma ao ambiente *Web*, inviabilizando que os recursos do ServiceBook sejam consumidos de forma nativa por aplicativos móveis (*Mobile*) no futuro, como também habilita a implementação de funcionalidades de *Progressive Web App* (Aplicação Web Progressiva) (PWA). Diante desses fatores, torna-se necessária a migração para um modelo de distribuição de dados via API REST, permitindo a separação total entre a lógica de negócio e a interface, ao mesmo tempo em que a padronização do código é elevada por meio de arquiteturas modernas, como o Monorepo.

3.3.2 Obsolescência Visual e Despadronização de Interface

Do ponto de vista da usabilidade e do UI, o sistema apresenta limitações decorrentes do desenvolvimento fragmentado realizado ao longo das diferentes edições da disciplina. Apesar do esforço de Luz (2024) em reorganizar e expandir as funcionalidades da plataforma, ainda são observadas inconsistências relacionadas à padronização visual da interface.

O uso do framework Materialize também contribuiu com as limitações relacionadas a personalização visual da interface. Por ser se tratar de um conjunto de componentes pré-estilizados, alterações em cores, espaçamentos e comportamentos específicos frequentemente exigiam sobrescrita excessiva de classes *Cascading Style Sheets* (Folhas de Estilo em Cascata) (CSS). Como consequência, diferentes módulos do sistema apresentam inconsistências perceptíveis em margens, tamanhos de fonte e disposição dos componentes, como botões e campos de formulário.

A obsolescência desse design legado pode ser observada nos painéis principais da aplicação. A Figura 6 apresenta a interface utilizada pelo cliente para o gerenciamento de solicitações de serviço, enquanto a Figura 7 ilustra o painel do profissional destinada a visualização dos anúncios disponíveis. Em ambas as interfaces, observa-se a ausência de padronização visual, além da utilização de componentes com baixa hierarquia visual e organização limitada dos elementos em tela.

Figura 6 – Interface legada: visão do cliente gerenciando solicitações.
Fonte: (LUZ, 2024).

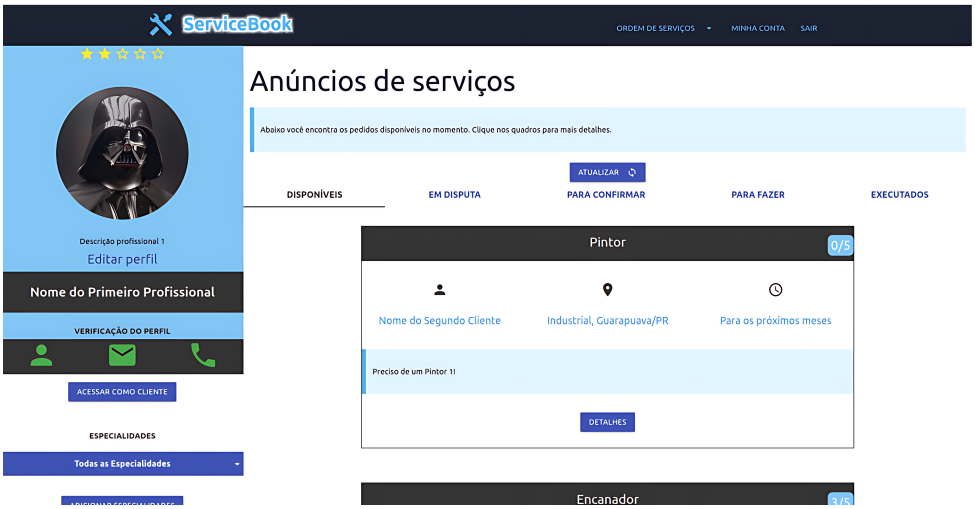

Figura 7 – Interface legada: visão do profissional buscando anúncios. Fonte: (LUZ, 2024).

3.3.3 Conclusão do Diagnóstico

O diagnóstico realizado evidencia que a arquitetura atual do ServiceBook apresenta limitações que dificultam a evolução da plataforma, especialmente em aspectos relacionados à escalabilidade, manutenção e experiência do usuário. Além disso, a integração de recursos modernos, como Inteligência Artificial e interfaces reativas, torna-se limitada dentro do modelo baseado em SSR.

Dessa forma, a migração de uma arquitetura monolítica baseada em *Server-Side Rendering* para uma estrutura desacoplada, fundamentada em uma API REST no *back-end* e em uma arquitetura baseada em componentes no *front-end*, apresenta-se como uma evolução necessária para garantir modernização da plataforma ServiceBook.

4 MATERIAIS E MÉTODOS

Este capítulo apresenta as tecnologias, ferramentas e metodologias que serão utilizadas no processo de modernização arquitetural e visual da plataforma *ServiceBook*. Como o projeto propõe uma reestruturação profunda, o detalhamento está dividido entre os artefatos tecnológicos selecionados (Materiais) e as práticas de engenharia e gestão adotadas para a execução (Métodos).

4.1 Materiais

Para a construção da nova versão do sistema, foi selecionada uma pilha tecnológica (*stack*) moderna, visando solucionar as limitações estruturais da arquitetura anterior. O *ServiceBook* legado operava com renderização no lado do servidor utilizando páginas JSP e armazenava seus dados em um banco MySQL. A nova proposta rompe com esse modelo, dividindo a aplicação em camadas independentes.

Na camada de apresentação (*front-end*), a interface gráfica será desenvolvida como uma *Single Page Application* (SPA) utilizando o *framework* Angular (Angular Team, 2026). Esta nova arquitetura substitui o padrão anterior baseado em navegação multipágina e na biblioteca Materialize CSS. Para a estilização visual e construção do *Design System*, será adotado o Tailwind CSS em conjunto com a biblioteca de componentes não estilizados (*headless*) *spartan/ng* (Spartan UI, 2026). Essa combinação permite a criação de componentes personalizados e acessíveis sem a necessidade de manter grandes arquivos CSS tradicionais. Além disso, a prototipação prévia das telas será realizada na ferramenta Stitch, possibilitando a validação visual com base nas histórias de usuário antes da etapa de codificação.

No *back-end*, a aplicação continuará a utilizar o ecossistema Spring Boot com Java (Spring Team, 2026), tecnologia consolidada nas versões anteriores, mas que agora atuará exclusivamente no fornecimento de uma API *RESTful*. Em relação à persistência de dados, o banco de dados MySQL que na versão legada operava na versão 5.7 será substituído pelo PostgreSQL. Essa transição é tecnicamente justificada pela necessidade de atualizar a infraestrutura para um padrão mais moderno e robusto de consistência de dados (*Atomicity, Consistency, Isolation, Durability* (Atomicidade, Consistência, Isolamento e Durabilidade) (ACID)). Além disso, há a questão do licenciamento: enquanto o MySQL 5.7 representa uma linhagem que sofreu mudanças de governança após a aquisição pela Oracle, o PostgreSQL mantém uma licença de código aberto irrestrita e comunitária (PostgresSql, 2026). A mudança ocorre sem impactos na lógica existente, uma vez que as bibliotecas de persistência do Spring (*Java Persistence API* (JPA) / Hibernate) garantem a compatibilidade e abstraem as diferenças entre os bancos. Durante o desenvolvimento local, esse banco de dados será executado isoladamente em contêineres através da plataforma Docker (Docker, 2026).

A infraestrutura de hospedagem e o controle de versão do código-fonte serão centralizados na plataforma GitHub. O versionamento será acompanhado pelo uso do GitHub Actions para *Continuous Integration* (Integração Contínua) (CI), responsável por automatizar a execução de testes a cada nova alteração enviada ao repositório (GitHub, 2026). A implantação (*deploy*) da aplicação, englobando tanto a API quanto a interface *web*, será realizada no serviço de computação em nuvem Render.

Por fim, o ambiente de desenvolvimento e codificação será a *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado) (IDE) Antigravity (Google Antigravity, 2026). A escolha justifica-se por tratar-se de um ecossistema projetado nativamente para o desenvolvimento assistido por Inteligência Artificial, oferecendo recursos avançados de orquestração de agentes. Além de sua integração nativa com o *Model Context Protocol* (Protocolo de Contexto de Modelo) (MCP) que permite padronizar a comunicação entre a IA e fontes de dados externas e heterogêneas, transformando-as em contexto acionável, a IDE provê capacidades de execução de planos de codificação (*workflows*) que permitem à IA não apenas sugerir trechos de código, mas realizar alterações estruturais coordenadas em múltiplos arquivos simultaneamente. O processo de escrita de código utilizará inicialmente as cotas gratuitas semanais de modelos de linguagem (LLMs) oferecidas pela IDE. Após o esgotamento dessas cotas, o projeto fará o uso de modelos gratuitos de alta performance em geração de código — como MiniMax e Qwen3 — disponibilizados através da integração da extensão OpenCode com a plataforma OpenRouter (Arena Leaderboard, 2026).

4.2 Metodologia

O gerenciamento do projeto será conduzido com base em uma adaptação da metodologia ágil *Scrum*. Como a codificação será realizada de forma individual, os papéis tradicionais do *framework* foram readequados para o contexto acadêmico: o orientador assumirá as funções de *Product Owner* (validação de requisitos, definição de regras de negócio e priorização) e *Scrum Master* (facilitação do processo e remoção de impedimentos), enquanto o aluno atuará como o *Developer* (executor técnico e orquestrador das ferramentas de IA).

O ciclo de trabalho será estruturado em Sprints com duração quinzenal. Esse intervalo é considerado ideal para a implementação de um conjunto viável de funcionalidades, permitindo a validação contínua do código gerado por Inteligência Artificial e a aplicação de ajustes sem comprometer o cronograma. Ao final de cada ciclo, será realizada uma Sprint Review, na qual o incremento do produto será apresentado ao Product Owner (orientador) para validação e feedback. Este encontro servirá, simultaneamente, como Sprint Planning para o ciclo seguinte, garantindo que as prioridades e o escopo da nova etapa estejam alinhados às necessidades do projeto e aos objetivos acadêmicos.

O acompanhamento da execução dessas tarefas será realizado por meio do *GitHub Projects*. Para a gestão visual do fluxo de trabalho, o quadro utilizará o modelo *Kanban* padrão da ferramenta, garantindo transparência no progresso através das seguintes etapas:

- ***To Do* (A Fazer):** Tarefas planejadas para a *Sprint* atual que ainda não tiveram sua execução iniciada e aguardam o refinamento de suas especificações.
- ***In Progress* (Em Progresso):** Atividades em desenvolvimento ativo, englobando a escrita de testes, a geração de código pela IA e os ajustes manuais.
- ***Review* (Revisão):** Tarefas finalizadas tecnicamente, mas que aguardam a validação dos testes automatizados ou a aprovação visual e arquitetural por parte do orientador.
- ***Done* (Concluídas):** Funcionalidades validadas, integradas ao sistema e prontas para entrega.

Quanto à organização do código-fonte, será adotada a estratégia de *monorepo* (repositório único), mantendo as aplicações *front-end* e *back-end* centralizadas. Com os arquivos no mesmo repositório, a Inteligência Artificial consegue analisar o contexto integral do sistema, compreendendo como as alterações na API impactam diretamente os componentes de interface.

No que diz respeito ao fluxo de integração e implantação, a metodologia de trabalho fará uso contínuo de esteiras automatizadas (CI/*Continuous Deployment* / *Continuous Delivery* (Implantação Contínua / Entrega Contínua) (CD)) orquestradas via *GitHub Actions*. O processo de desenvolvimento exigirá que, a cada novo *commit* submetido ao repositório principal, uma *pipeline* seja acionada para executar os testes automatizados. O código apenas avançará para a etapa de implantação (*deploy*) caso os testes automatizados validem com sucesso os critérios de aceitação definidos para as referidas funcionalidades. No ambiente local, o fluxo de testes e execução adotará a conteinerização para garantir que o banco de dados, a API e a interface operem de forma isolada.

4.3 Desenvolvimento Agêntico e SDD

A principal inovação metodológica deste projeto reside na aplicação do Desenvolvimento Agêntico, no qual ferramentas de Inteligência Artificial autônoma não atuam apenas como assistentes de código, mas como co-pilotos integrados ao ciclo de vida do *software*. Na tentativa de evitar que os agentes de IA cometam erros de arquitetura ou sofram de alucinações (geração de código inventado ou fora de contexto), o projeto adotará a abordagem *Specification-Driven Development* (Desenvolvimento Orientado a Especificações).

Nesse modelo, o fluxo de trabalho é estritamente documentado e dividido em duas etapas fundamentais: uma visão Macro, focada no planejamento e configuração, e uma visão Micro, focada na implementação iterativa.

4.3.1 Visão Macro: Planejamento, Especificação e *Harness*

A etapa Macro antecede qualquer codificação de funcionalidades. Ela é responsável por traduzir as necessidades de negócio em contratos técnicos rigorosos que alimentarão o contexto da IA. Esse processo inicia-se com a elaboração do documento de requisitos do produto.

O *Product Requirements Document* (Documento de Requisitos do Produto) (PRD) será escrito utilizando uma linguagem de alto nível, voltada para o negócio, sem focar em implementação técnica. Ele conterá a visão geral do sistema, o objetivo central e o mapeamento dos atores e suas respectivas permissões. O mapeamento dos requisitos será estruturado em duas práticas fundamentais:

- **Histórias de Usuário e Critérios de Aceitação:** As funcionalidades serão descritas no formato clássico (“Como [ator], quero [ação] para que [benefício]”). Cada história será acompanhada de seus respectivos Critérios de Aceitação, redigidos sob a sintaxe do *Behavior-Driven Development* (Desenvolvimento Orientado a Comportamento) (BDD) — estruturados no formato “Dado que [contexto], Quando [ação], Então [resultado]”. Esses critérios atuarão como regras estritas de validação daquela funcionalidade. Eles são vitais para o projeto, pois servirão de roteiro exato para a Inteligência Artificial gerar os testes automatizados na fase do TDD.

- **Gestão de Escopo e Priorização *Must have, Should have, Could have, Won’t have* (Tem que ter, Deveria ter, Poderia ter, Não vai ter) (MoSCoW):** Para organizar o *backlog* e definir o que será de fato desenvolvido, será adotado o método MoSCoW (*Must have, Should have, Could have, Won’t have*). Essa técnica tornará o escopo do projeto flexível, concentrando os esforços primários nos itens essenciais (*Must have*). Um fator crítico para o sucesso do Desenvolvimento Agêntico é a definição rigorosa dos itens “*Won’t have*” (Fora de Escopo). Esse mapeamento impõe limites cognitivos ao modelo de IA, impedindo que o agente “alucine” e tente implementar recursos não planejados.

Por fim, o PRD também documentará as restrições sistêmicas, regras de negócio gerais (*Constraints*) e os requisitos de qualidade.

Com o negócio mapeado, o desenvolvedor traduzirá o PRD para o *Software Design Document* (Documento de Design de Software), utilizando uma linguagem estritamente técnica. Este documento funcionará como o “manual de instruções arquiteturais” da IA. Ele detalhará a arquitetura em repositório único (*Monorepo*), a lista de integrações via MCP e a *stack* tecnoló-

gica com a indicação exata das versões das ferramentas, *frameworks* e bibliotecas e os *links* para as documentações oficiais atualizadas. O SDD conterá também a modelagem de dados descrita em diagramas estruturais textuais (*Mermaid*), a estrutura de diretórios, as diretrizes de segurança, o padrão das rotas (*REST Guidelines*), o gerenciamento de variáveis de ambiente e os contratos globais, como *Data Transfer Objects* (*Data Transfer Object* (Objeto de Transferência de Dados) (DTO)s) e interfaces.

Ainda na visão Macro, o ambiente de desenvolvimento (IDE *Antigravity*) será configurado com um *Harness*. Na engenharia de *software* assistida por Inteligência Artificial, o *Harness* (que pode ser compreendido como um arcabouço de controle) é o ecossistema que envolve e restringe o modelo de linguagem. Como as IAs generativas são probabilísticas e livres por natureza, o *Harness* atua impondo limites (*guardrails*) e fornecendo contexto, transformando uma IA genérica em um agente de desenvolvimento previsível e seguro.

Essa configuração será estruturada em três pilares funcionais:

- ***Skills* (Habilidades):** Capacitações específicas instaladas no ambiente para que a IA compreenda profundamente e consiga interagir com as tecnologias escolhidas (como, por exemplo, a habilidade de executar comandos no terminal ou consultar a documentação oficial atualizada do Angular e do *Spring Boot*).

- ***Rules* (Regras):** Diretrizes rígidas de comportamento e arquitetura. Elas servem para padronizar o código gerado, forçar a adoção dos padrões do projeto e, principalmente, evitar alucinações (impedindo que a IA invente bibliotecas ou fuja do escopo estabelecido).

- ***Workflows* (Fluxos Automatizados):** Roteiros de execução determinísticos que o agente deve seguir passo a passo, garantindo que o processo não dependa de *prompts* aleatórios do usuário. O primeiro *workflow* a ser executado, por exemplo, será o de *scaffolding* (construção estrutural), no qual a IA lerá as instruções do documento SDD para gerar a arquitetura base, os diretórios e os arquivos iniciais do repositório, com validação pelo desenvolvedor de cada etapa da construção estrutural.

4.3.2 Visão Micro: Implementação Iterativa e Automação

Com a base estrutural do sistema devidamente configurada, o desenvolvimento entrará na fase Micro. Esta etapa consiste em um ciclo repetitivo focado em implementar cada história de usuário (validada previamente pelo método MoSCoW), utilizando as automações configuradas na fase Macro.

O ciclo de desenvolvimento de cada nova funcionalidade ocorrerá rigorosamente através dos seguintes passos:

1. **Recuperação de Contexto e Versionamento:** O ciclo inicia-se com o *workflow* assumindo a gestão do *GitFlow*, criando automaticamente uma ramificação específica para a nova funcionalidade (*Feature Branch*). Em seguida, o desenvolvedor informará o identificador Identificador (ID) da História de Usuário presente no *Kanban* (*GitHub Projects*). Utilizando o protocolo MCP, o agente de IA acessará o repositório, fará a leitura dos Critérios de Aceitação no PRD e, caso exista, buscará o protótipo de tela correspondente na ferramenta de prototipação (*Stitch*).

2. **Quebra de Tarefas:** A Inteligência Artificial analisará a demanda e a desmembrará em tarefas técnicas menores, separando logicamente as alterações que ocorrerão no banco de dados, na lógica de negócio da API (*back-end*), na construção visual da interface e nas regras de apresentação (*front-end*).

3. **TDD:** Para as camadas que exigem regras de negócio, a codificação seguirá estritamente o Desenvolvimento Orientado por Testes. Guiado pelos critérios de aceitação, o agente gerará primeiramente os esqueletos dos testes automatizados sem a lógica de negócio (fazendo com que os testes falhem intencionalmente).

4. **Implementação e Refatoração:** Utilizando suas *Skills* ativas, a IA implementará o código real com o objetivo exclusivo de fazer com que os testes automatizados passem na validação. Confirmada a funcionalidade (testes com *status* verde), o agente executará a etapa de refatoração estrutural para aplicar boas práticas, remover redundâncias de código e otimizar o desempenho.

5. **Governança e Autocorreção Agêntica (*Husky*):** Ao finalizar a codificação, o agente tentará registrar as alterações no controle de versão (*Commit*). Neste exato momento, o *Harness* entrará em ação através dos gatilhos de pré-*commit* do *Husky* e do *Lint-Staged*. O sistema rodará as ferramentas de análise estática (como *ESLint*, *Prettier* e *Checkstyle*). Se o código gerado contiver erros de formatação ou violações de regras arquiteturais, o *Husky* bloqueará o *commit* e retornará o erro. A IA, de forma autônoma, lerá o erro no terminal, corrigirá seu próprio código e tentará realizar o *commit* novamente, em um ciclo de autocorreção sem intervenção humana.

6. **Revisão Humana (*Code Review*):** Somente após o código passar pelos testes do TDD e pela barreira do *Husky*, o desenvolvedor assumirá o papel de revisor. Estando a implementação de acordo com os padrões visuais e de engenharia exigidos, o desenvolvedor aprovará a solução e abrirá um Pedido de Mesclagem *Pull Request* (Pedido de Mesclagem) (PR). Caso sejam identificadas falhas lógicas ou desvios arquiteturais, o desenvolvedor enviará um *feedback* textual apontando o erro ao agente de IA, que processará a instrução, refatorará o código e reiniciará o ciclo de validação. Esse laço de *feedback* garante a evolução contínua da funcionalidade, mantendo a responsabilidade de escrita com a ferramenta sob a orquestração humana.

7. **CI:** No momento em que o PR é aberto, a esteira do *GitHub Actions* executa a suíte completa de testes em ambiente de nuvem para garantir que a nova funcionalidade não cause regressões. Sendo aprovado nos critérios técnicos e na revisão humana, o código é mesclado na *branch develop*. Esta ramificação funciona como um ambiente de integração e estabilização, onde múltiplas funcionalidades são consolidadas e testadas em conjunto antes de comporem uma versão oficial do sistema.

8. **CD e *Release*:** Após a validação de um conjunto de funcionalidades estáveis na *develop*, o desenvolvedor realiza o processo de *release*, mesclando as alterações na *branch* principal (*main*). A plataforma *Render* monitora exclusivamente a *main*, disparando automaticamente o fluxo de *deploy* da API e da SPA sempre que uma nova versão estável é detectada. Este passo encerra o ciclo de desenvolvimento agêntico, disponibilizando a versão atualizada da plataforma no ambiente de produção.

5 ANÁLISE DO SISTEMA

Este capítulo detalha a análise de requisitos e a modelagem do escopo para o projeto de modernização do *ServiceBook*. Em alinhamento com a metodologia de SDD adotada neste trabalho, esta análise atua como a materialização da “Visão Macro” do projeto. O conteúdo aqui estruturado desempenha a função de um PRD, servindo não apenas como documentação humana, mas como o contrato de escopo rigoroso que alimentará o contexto e imporá limites cognitivos aos agentes de Inteligência Artificial durante a fase de codificação.

A solução foca na quebra do acoplamento da arquitetura legada (baseada em JSP) e na reestruturação do sistema sob o modelo de uma API *Restful* em *Spring Boot* e uma SPA em Angular, introduzindo como f de negócio o uso de LLMs para a implementação de novas funcionalidades.

5.1 Perfis de Usuários (Atores)

O ecossistema modernizado do *ServiceBook* mapeia os seguintes perfis principais para a interação com a plataforma:

- **Cliente (Contratante):** Usuário final que busca por serviços. Seu principal objetivo é encontrar profissionais qualificados de forma ágil, navegar em portfólios fluidos e tomar decisões baseadas em avaliações de reputação sintetizadas por Inteligência Artificial.
- **Profissional (Prestador de Serviço):** Usuário que oferta seus serviços na plataforma. Necessita expor seu portfólio e gerenciar seus dados profissionais com segurança e praticidade.
- **Sistema (Infraestrutura/IA):** Ator lógico responsável por garantir o isolamento arquitetural via autenticação, executar rotinas em *background* e orquestrar a comunicação externa com a API de Inteligência Artificial para a geração de resumos.

5.2 Tarefas Técnicas

Antes de detalhar as funcionalidades perceptíveis aos usuários finais, é necessário definir as tarefas técnicas do projeto. Estas atividades englobam o trabalho de infraestrutura e refatoração que ocorre nos bastidores da aplicação. Embora não sejam vistas diretamente pelo cliente, elas constituem a base estrutural indispensável para viabilizar o desenvolvimento da nova versão do *ServiceBook*:

- **T01 - Monorepo e Integração Contínua (CI):** Organização do projeto em um repositório único (*monorepo*) e configuração do *GitHub Actions* para automatizar os processos de compilação e testes, garantindo a verificação do código a cada nova atualização.

- **T02 - Transição para API RESTful e JWT:** Remoção das páginas JSP do *back-end* em *Spring Boot*, alterando o sistema para comunicar-se apenas por meio de dados em formato *JavaScript Object Notation* (Notação de Objetos JavaScript) (JSON) e implementando a autenticação baseada em *tokens* (JWT).
- **T03 - Construção da Interface Base no Angular:** Criação de componentes visuais reutilizáveis para o *front-end*, utilizando as ferramentas *Tailwind* CSS e *spartan/ng* para estabelecer um padrão estético uniforme e facilitar a criação das novas telas.
- **T04 - Containerização com Docker:** Configuração do sistema em *containers* isolados para assegurar que a aplicação funcione exatamente com o mesmo comportamento, tanto na máquina local de desenvolvimento quanto no servidor final.
- **T05 - Configuração do Ambiente de Desenvolvimento com IA:** Estruturação do ecossistema de ferramentas de Inteligência Artificial que apoiará a escrita do código. Isso engloba a configuração do ambiente de desenvolvimento (*IDE*) para integração com modelos de linguagem (*LLMs*, como o OpenCode), a definição das regras de comportamento (*Rules*) e habilidades (*Skills*) dos agentes autônomos, além da implementação de *workflows*. Essa preparação é fundamental para que a IA consiga ler os documentos de especificação do projeto e auxiliar na programação de forma precisa e contextualizada.

6 PROJETO DO SISTEMA

Neste capítulo, são detalhadas as decisões arquiteturais, a organização estrutural e os padrões visuais e tecnológicos definidos para a modernização do *ServiceBook*. Ao consolidar a estruturação do *back-end*, as diretrizes de *front-end* e a prototipação das interfaces, estabelece-se o alicerce técnico necessário para a construção da nova plataforma.

Conforme abordado na fundamentação metodológica, essas especificações arquiteturais e de *design* formam a base prática de contexto para o SDD. O conjunto de definições apresentadas a seguir atua não apenas como um guia para os desenvolvedores, mas fornece as instruções e restrições fundamentais consumidas pelos agentes de Inteligência Artificial, garantindo governança técnica, previsibilidade e alinhamento estrutural durante todo o processo de desenvolvimento assistido por IA.

6.1 Back-end

Nesta seção, são apresentadas as principais alterações na camada de servidor para a execução do processo de modernização do *ServiceBook*, adaptando a arquitetura para o fluxo de trabalho com agentes de IA. A mudança estrutural primária consiste na transição do modelo monolítico para a oferta de dados via API REST.

Na versão legada do *ServiceBook*, os controladores Spring MVC assumiam a dupla responsabilidade de processar as regras de negócio e renderizar as páginas JSP diretamente no servidor. A Listagem 6.1 apresenta um exemplo simplificado do controlador responsável pela exibição da tela de gerenciamento de profissionais da empresa.

```
1  @GetMapping()
2  @RolesAllowed({RoleType.COMPANY})
3  public ModelAndView showProfessionals() throws Exception {
4
5      User company = this.getCompany();
6
7      ModelAndView mv =
8          new ModelAndView("company/new-professional");
9
10     List<User> professionals =
11         userService.findProfessionalsNotExist();
12
13     mv.addObject("professionals", professionals);
14
15     return mv;
16 }
```

Listing 6.1 – Exemplo de controlador legado utilizando JSP

Nesse modelo, o controlador retorna diretamente uma página JSP renderizada no servidor, utilizando o objeto *ModelAndView* para transferir os dados para a camada de visualização. Essa abordagem cria forte acoplamento entre interface e back-end, dificultando a reutilização da aplicação por diferentes clientes.

Na nova arquitetura proposta neste trabalho, os controladores passarão a operar exclusivamente como fornecedores de dados através de uma API REST. Dessa forma, ao invés de retornar páginas JSP, os endpoints retornarão DTOs serializados automaticamente em JSON pelo Spring Boot.

A Listagem 6.2 apresenta um exemplo simplificado de como o mesmo controlador poderá ser adaptado para o novo modelo arquitetural baseado em API REST.

```
1  @GetMapping("/api/professionals")
2  public ResponseEntity<List<ProfessionalDTO>> listProfessionals() {
3
4      List<ProfessionalDTO> professionals =
5          userService.findProfessionalsDTO();
6
7      return ResponseEntity.ok(professionals);
8  }
```

Listing 6.2 – Exemplo de controlador REST utilizando DTOs

Nesse novo modelo, o Spring Boot realiza automaticamente a serialização dos objetos DTO para o formato JSON, permitindo desacoplamento entre a interface Angular e o servidor. A Listagem 6.3 apresenta um exemplo simplificado da estrutura JSON retornada pela API REST.

```
1  {
2    "id": 1,
3    "name": "João Silva",
4    "category": "Eletricista",
5    "rating": 4.8
6  }
```

Listing 6.3 – Exemplo de resposta JSON da API

Essa abordagem permite maior reutilização da camada de back-end, possibilitando integração com aplicações SPA, aplicativos móveis e outros serviços externos de forma padronizada. A API será documentada utilizando Swagger/OpenAPI, permitindo visualização dos endpoints e contratos de comunicação entre *front-end* e *back-end*. Além disso, a transição para este modelo *stateless* pavimenta o caminho para a implementação de protocolos de autenticação modernos, como o JWT, garantindo um tráfego de dados mais seguro e independente da renderização de telas.

6.2 Front-end

Para o desenvolvimento da interface modernizada, será adotado o *framework* Angular em sua versão mais recente, em conjunto com a linguagem TypeScript e a ferramenta Tailwind. A seguir, será apresentado o novo *design system* elaborado para a construção das principais telas da aplicação, estabelecendo um comparativo visual entre o sistema legado e a nova proposta. Ressalta-se que as interfaces exibidas nesta etapa consistem em protótipos e estão sujeitas a refinamentos e ajustes ao longo do ciclo de desenvolvimento.

6.2.1 Paleta de Cores

A paleta de cores foi selecionada com o objetivo de transmitir uma percepção de produto atual, conferindo uma aparência *premium* e uma identidade visual alinhada a plataformas tecnológicas modernas.

Para a construção do *Design System*, foram definidas cores base para os elementos principais da interface. A Figura 8 ilustra não apenas esses valores hexadecimais centrais, mas também as suas respectivas escalas tonais (variações de luminosidade e saturação). Essa amplitude é essencial para gerenciar estados de interação (como efeitos de sobreposição (*hover*) e ativação (*active*)) e para assegurar o contraste adequado em termos de acessibilidade. A disponibilidade da escala completa permite, por exemplo, aplicar uma tonalidade suave para o fundo de um componente de alerta e uma tonalidade escura da mesma cor para a tipografia, garantindo a legibilidade. Esta abordagem, estruturada em múltiplas tonalidades (geralmente indexadas de 50 a 900), segue a arquitetura padrão do *framework Tailwind* CSS adotado no projeto:

- **Background do sistema:** Midnight Navy (`#0F172A`). Tom escuro profundo para reduzir o cansaço visual.
- **Texto e Ícones:** Ghost White (`#F8FAFC`). Utilizado para garantir alto contraste sobre os fundos escuros.
- **Elementos de destaque e ações:** Sky Blue (`#0EA5E9`). Cor primária de atenção, aplicada em botões principais e *links*.
- **Componentes secundários:** Slate Gray (`#334155`). Empregado em bordas, divisórias e botões de menor prioridade na hierarquia visual.
- **Cores de texto**: Ghost White sobre fundos escuros e Sky Blue para links e destaques.

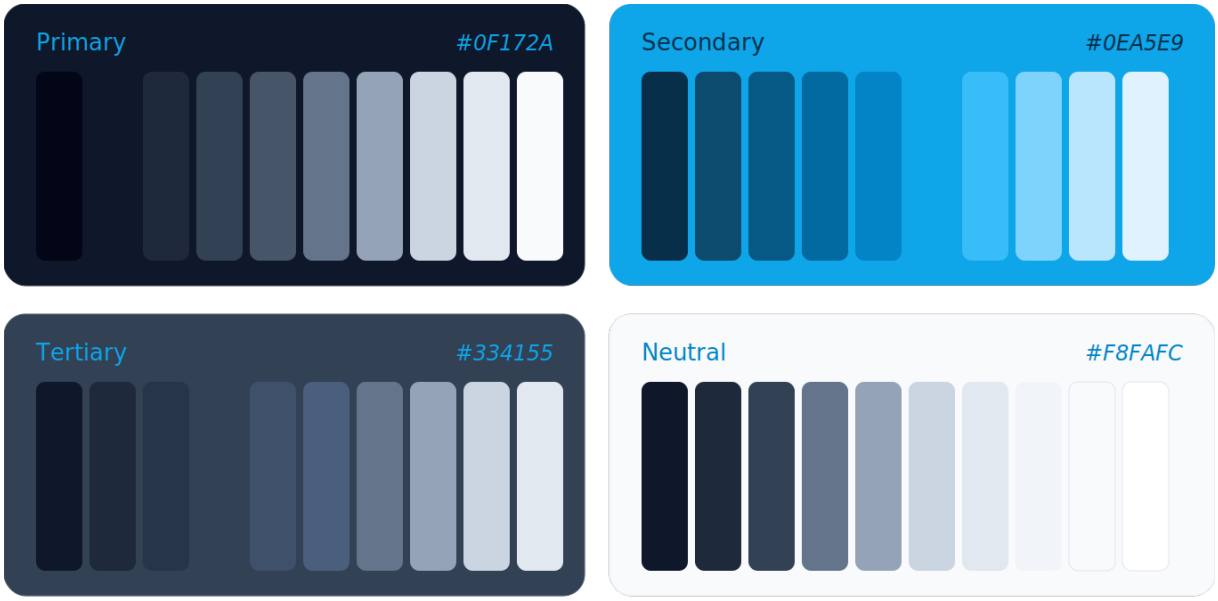

Figura 8 – Paleta do ServiceBook
Fonte: Autor (2026).

6.2.2 Tipografia

A tipografia escolhida para o novo design é a fonte **Manrope** apresentada na figura 9, uma família tipográfica sem serifa desenvolvida para interfaces digitais modernas. Sua escolha ocorreu devido às características de alta legibilidade, traços geométricos equilibrados e boa adaptação a diferentes resoluções e tamanhos de tela, fatores importantes para aplicações web responsivas.

Além do aspecto visual minimalista e contemporâneo, a fonte apresenta excelente hierarquia tipográfica, permitindo melhor distinção entre títulos, subtítulos e conteúdos textuais. Outro fator considerado foi sua ampla utilização em sistemas e plataformas digitais modernas, especialmente em aplicações *Software as a Service* (Software como Serviço) (SaaS) e interfaces voltadas à experiência do usuário.

Dessa forma, a utilização da fonte Manrope contribui para a construção de uma identidade visual mais profissional, acessível e consistente, proporcionando melhor experiência de leitura e navegação aos usuários da plataforma.

Esta fonte também é utilizada em alguns elementos da interface do ClickUp, plataforma de gerenciamento de projetos e produtividade.

Figura 9 – Fonte Manrope utilizada no ServiceBook
Fonte: Autor (2026).

6.2.3 Logotipo

A nova versão do logotipo passou por um processo de *redesign* sob uma perspectiva mais moderna e minimalista. O conceito original, que incorpora ferramentas em sua composição, foi preservado. No entanto, os elementos visuais foram estrategicamente arranjados em formato de "S"para fazer alusão à inicial da plataforma. Além disso, a nova paleta de cores foi ajustada para garantir contraste e integração harmoniosa com o tema escuro (*dark mode*) adotado pelo novo *design system*.

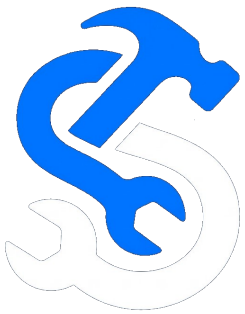

Figura 10 – Logotipo do ServiceBook
Fonte: Autor (2026).

6.3 Prototipação de Telas

Para a etapa de prototipação das interfaces, foi utilizada a ferramenta de design **Stitch**. Nesta seção, são analisadas as principais telas desenvolvidas para a nova versão do sistema, estabelecendo uma comparação entre a interface legada e a proposta visual modernizada.

Durante o processo de desenvolvimento das interfaces, foi adotada uma abordagem baseada em componentes reutilizáveis, permitindo maior padronização visual, facilidade de manutenção e agilidade na realização de alterações nos elementos da interface.

6.3.1 Login

Na Figura 12, é apresentado o novo layout da tela de login, enquanto a Figura 11 exibe a versão legada da interface. Ao comparar ambas as versões, observa-se que a interface antiga apresenta baixa hierarquia visual, excesso de elementos concorrendo pela atenção do usuário e pouca ênfase nos campos destinados ao processo de autenticação.

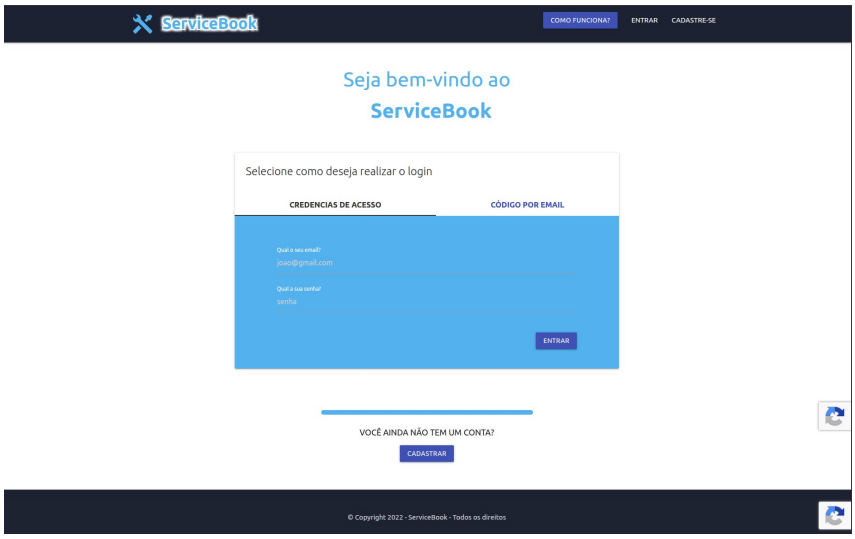

Figura 11 – Tela de login da versão legada
Fonte: ServiceBook (2024).

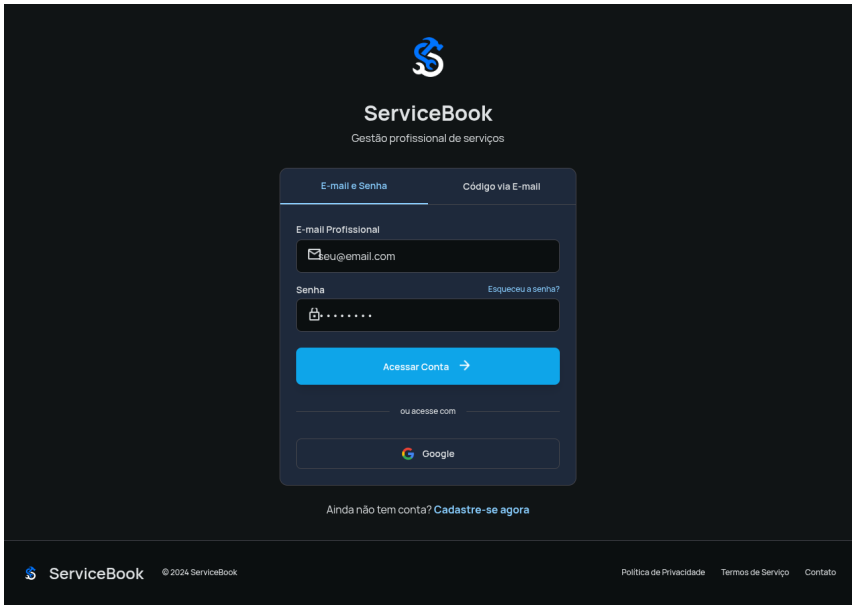

Figura 12 – Tela de login da nova versão
Fonte: Autor (2026).

Na nova versão, o layout foi redesenhado com foco na clareza visual e na centralização das informações essenciais para autenticação. Os campos de entrada passaram a ser destacados visualmente por meio de um componente do tipo *card*, contribuindo para o isolamento do formulário em relação aos demais elementos da interface e reduzindo distrações durante a interação. Além disso, foram incorporados ícones aos campos de entrada com o objetivo de reforçar semanticamente as informações solicitadas, facilitando o reconhecimento rápido e a compreensão dos dados requeridos pelo sistema.

Outro ponto de melhoria consiste na simplificação estrutural da tela, especialmente com a remoção da barra de navegação nesta etapa do fluxo. Essa decisão reduz a presença de elementos concorrentes e direciona o foco do usuário exclusivamente para a autenticação. A identidade visual do sistema também foi reforçada por meio da centralização da logomarca e do nome da aplicação acima do formulário, contribuindo para o reconhecimento da marca e para a consistência visual da interface.

Dessa forma, o redesenho busca aprimorar a usabilidade da tela de login, reduzindo ruídos visuais, fortalecendo a hierarquia da informação e tornando o processo de autenticação mais intuitivo e objetivo.

6.3.2 Solicitações do Cliente

Na Figura 13, observa-se que a interface legada apresenta elementos com elevado contraste de cores, porém sem uma padronização visual consistente, comprometendo a uniformidade da interface. Como exemplo, destacam-se os ícones verdes na seção de verificação de perfil. Embora tenham a importante função de indicar que os dados do usuário (documento, e-mail e celular) foram devidamente validados pelo sistema, a aplicação de uma cor vibrante

sobre o fundo escuro destoa da paleta do restante da página. Além disso, o uso de ícones que comumente sugerem ações interativas (como o símbolo de um telefone para ligações) para representar indicadores estáticos de status pode confundir o usuário.

Essa ausência de consistência visual pode dificultar a identificação e localização dos elementos da interface, impactando negativamente a experiência de navegação do usuário.

Já na Figura 14, observa-se a aplicação consistente de um *design system* em todos os componentes da interface. Além disso, foram incorporadas *tags* informativas em cada solicitação, contribuindo para melhor organização e compreensão das informações apresentadas.

A principal diferença entre as duas versões consiste na adoção de uma estrutura baseada em *cards*, proporcionando melhor aproveitamento do espaço em tela e alinhando a interface a padrões amplamente utilizados em plataformas de *e-commerce*.

A nova interface também oferece maior legibilidade e facilidade de navegação, permitindo que o usuário localize informações de forma mais rápida por meio de imagens, títulos e filtros, reduzindo a carga cognitiva durante a interação.

Figura 13 – Interface legada: visão do cliente gerenciando solicitações.
Fonte: (LUZ, 2024).

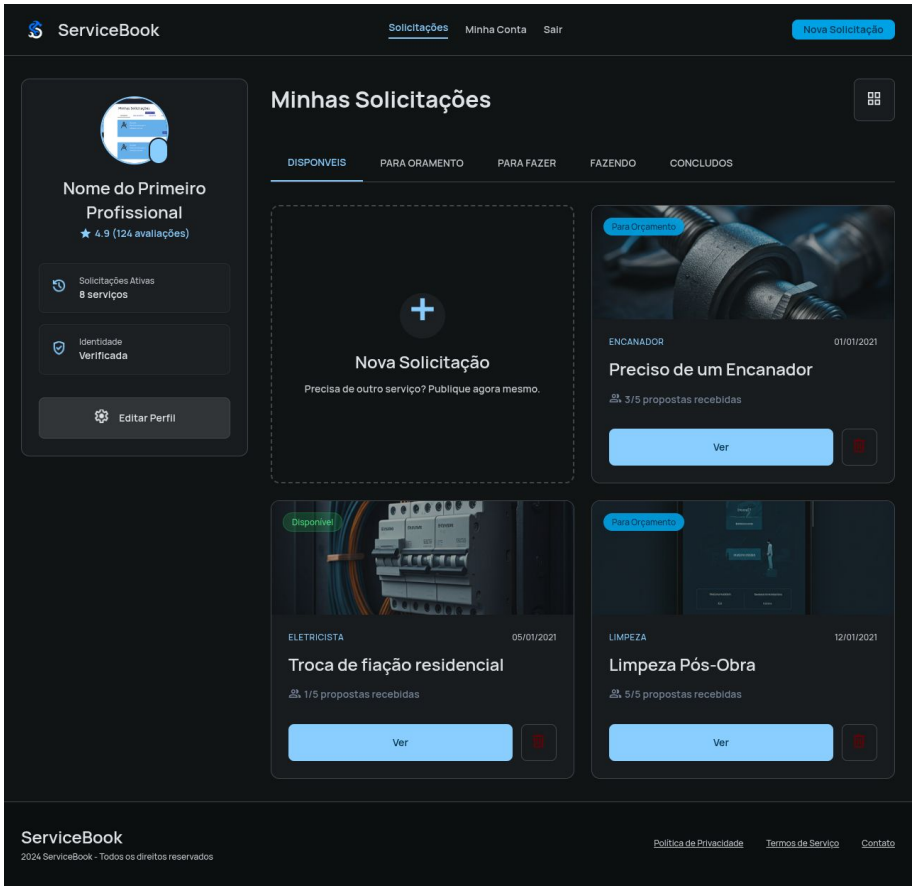

Figura 14 – Interface atualizada: visão do cliente gerenciando solicitações.
Fonte: Autor (2026).

6.3.3 Anúncios de Serviços

Na Figura 15, é apresentada a interface legada da listagem de anúncios visualizada pelo prestador de serviços, enquanto a Figura 16 exibe a nova versão da interface com a aplicação do *design system* desenvolvido para o sistema.

Na versão legada, observa-se um baixo aproveitamento do espaço disponível em tela, uma vez que cada anúncio ocupa praticamente toda a largura da interface. Como consequência, há necessidade de rolagem excessiva para visualização das demais publicações, comprometendo a experiência de navegação do usuário. Além disso, percebe-se que poucas informações são disponibilizadas previamente nos anúncios, tornando necessário acessar individualmente cada publicação para obter detalhes adicionais sobre o serviço ofertado. Consequentemente, o usuário precisa retornar constantemente à listagem caso o serviço visualizado não atenda às suas expectativas.

Na nova interface, foi adotada uma estrutura baseada em *cards*, permitindo melhor organização visual e maior aproveitamento do espaço disponível. Essa abordagem reduz a necessidade de rolagem constante e melhora significativamente a experiência de navegação. Outro aspecto relevante consiste na utilização de imagens como forma de *preview* do serviço anun-

ciado, permitindo que o usuário obtenha uma compreensão inicial do trabalho oferecido sem a necessidade de acessar imediatamente a publicação completa.

Dessa forma, a nova interface proporciona maior legibilidade, organização das informações e eficiência na busca por anúncios, tornando a navegação mais intuitiva e objetiva.

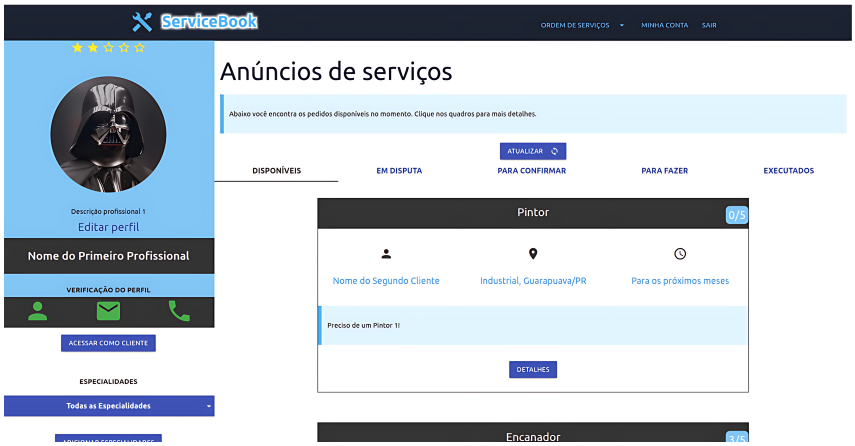

Figura 15 – Interface legada: visão do profissional buscando anúncios.
Fonte: (LUZ, 2024).

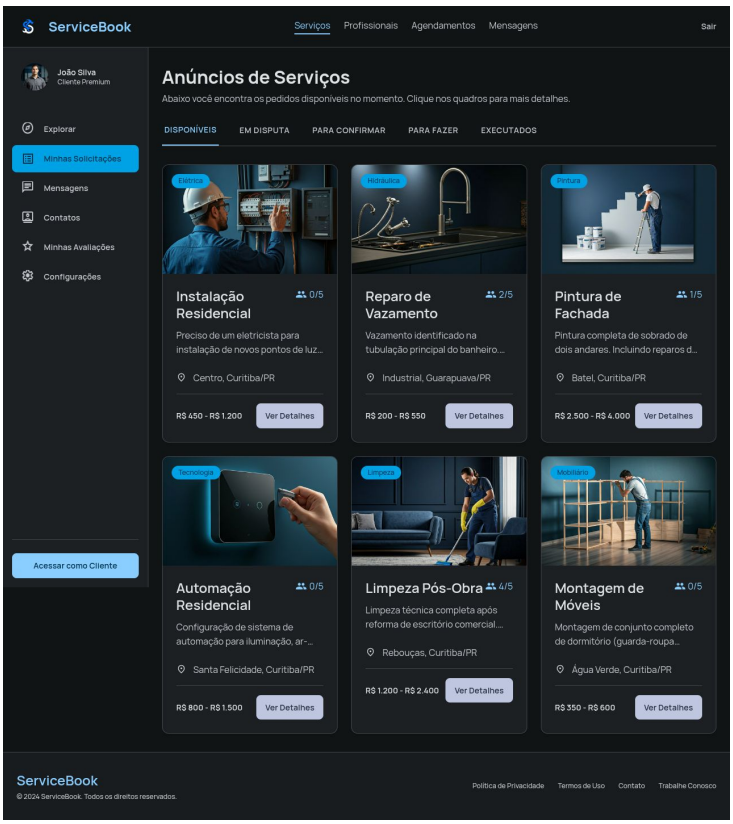

Figura 16 – Interface atualizada: visão do profissional buscando anúncios.
Fonte: Autor (2026).

6.3.4 Avaliação

Na Figura 17, é apresentada a interface legada da área de avaliações do prestador de serviços. Nessa versão, as avaliações eram exibidas apenas no formato tradicional de comentários individuais, exigindo que o usuário analisasse manualmente cada avaliação para compreender melhor a qualidade dos serviços prestados.

Já na Figura 18, é apresentada a nova interface da área de avaliações, que passa a contar com um resumo gerado por IA, contendo os principais pontos positivos e negativos mencionados pelos clientes sobre o prestador de serviços.

O resumo gerado pela IA tem como objetivo reduzir o tempo necessário para análise das avaliações, permitindo que o usuário obtenha rapidamente uma visão geral sobre a reputação do profissional sem a necessidade de ler individualmente todos os comentários disponíveis.

Além do resumo automatizado, a nova interface também exibe o portfólio de serviços realizados pelo prestador, informações adicionais sobre o profissional e um botão para solicitação de serviços, centralizando os principais recursos relacionados ao perfil em uma única tela.

Dessa forma, a nova interface proporciona melhor organização das informações, maior praticidade na análise das avaliações e uma experiência de navegação mais eficiente e objetiva.

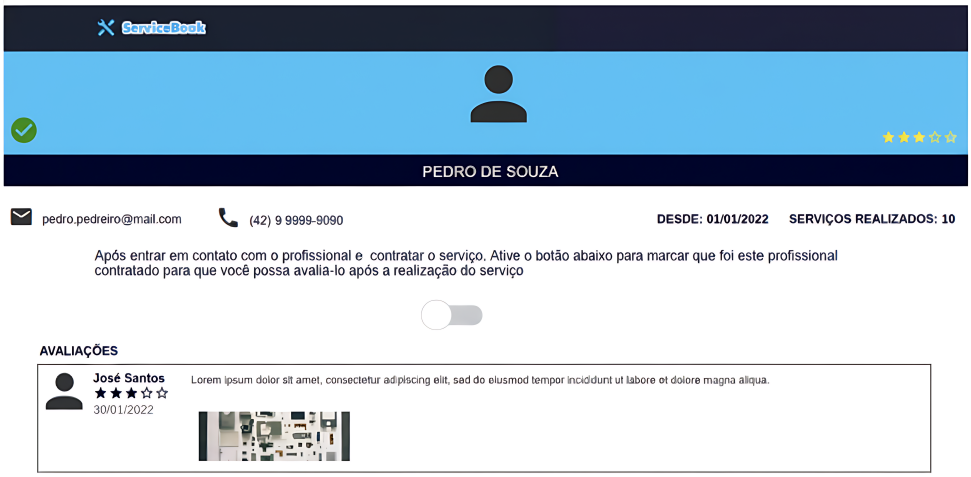

Figura 17 – Interface legada: avaliação de serviços.
Fonte: (LUZ, 2024).

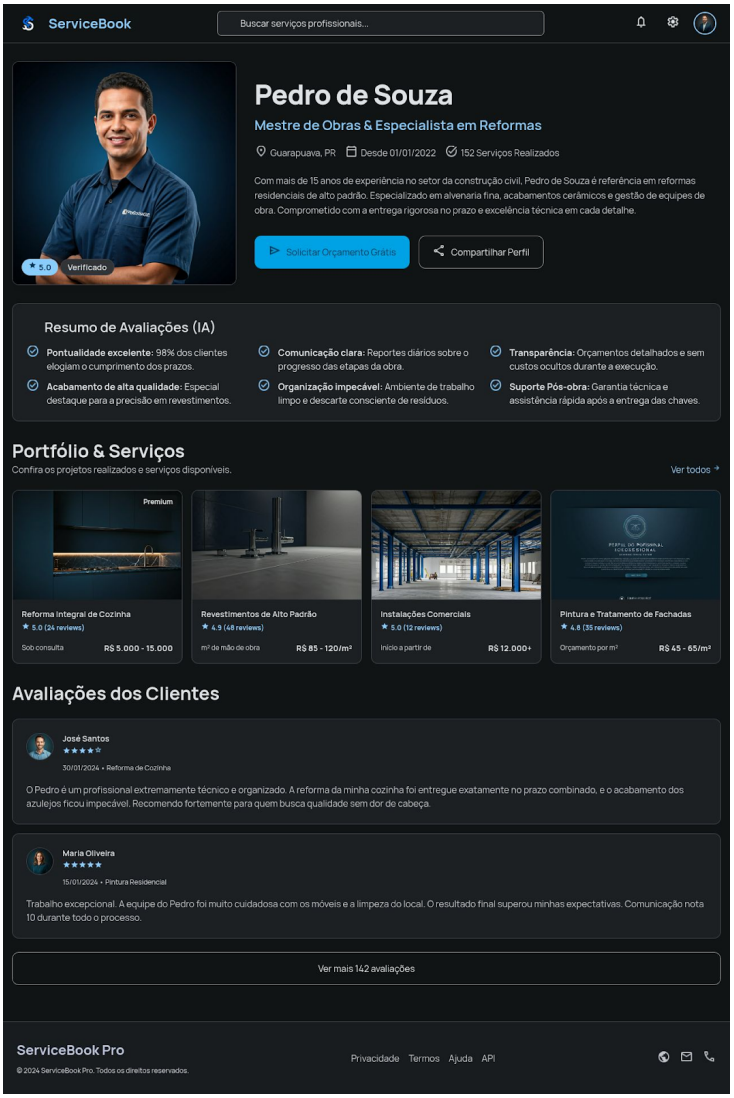

Figura 18 – Interface atualizada: avaliação de serviços.

Fonte: Autor (2026).

7 CONCLUSÃO

O presente trabalho apresenta o projeto de modernização arquitetural da plataforma ServiceBook, que atualmente possui uma arquitetura monolítica baseada em JSP com forte acoplamento entre interface e servidor. A estrutura legada dificulta a manutenção, a evolução tecnológica e a escalabilidade da aplicação, além de limitar a experiência dos usuários finais.

Como solução, o projeto implementará uma nova arquitetura desacoplada baseada em API REST no *back-end* e uma SPA desenvolvida em Angular no *front-end*. A nova abordagem permitirá separar as responsabilidades entre as camadas da aplicação, proporcionando maior flexibilidade, reutilização de componentes e facilidade de manutenção.

Além da modernização estrutural, o projeto também contemplará a criação de um novo *Design System*, com foco em padronização visual, responsividade e melhoria da experiência do usuário. As novas interfaces buscarão aproximar a plataforma dos padrões visuais presentes em aplicações modernas de *marketplace* digital.

Outro diferencial do projeto será a integração de Inteligência Artificial Generativa para sumarização automática das avaliações dos profissionais, agregando valor ao processo de tomada de decisão dos clientes dentro da plataforma.

No contexto metodológico, o trabalho aplicará conceitos modernos de engenharia de software, como SDD, utilização de agentes de Inteligência Artificial no processo de desenvolvimento, documentação arquitetural baseada em PRD e *Software Design Document*, além de práticas ágeis e integração contínua.

Dessa forma, conclui-se que a modernização projetada para o ServiceBook não representará apenas uma atualização tecnológica, mas também uma evolução arquitetural alinhada aos padrões contemporâneos do desenvolvimento web moderno, tornando a plataforma mais preparada para futuras expansões e novas funcionalidades.

REFERÊNCIAS

Angular Team. **Angular Documentation**. 2026. https://angular.dev. Acesso em: 18 maio 2026.

Arena Leaderboard. **Arena Leaderboard**. 2026. https://arena.ai/leaderboard. Acesso em: 20 de maio 2026.

Docker. **Docker Documentation**. 2026. https://docs.docker.com/. Acesso em: 20 de maio 2026.

ESTATíSTICA, I. B. de Geografia e. **PIB fecha 2024 em 3,4 e registra maior crescimento desde 2021**. 2025. Disponível em: https://agenciagov.ebc.com.br/noticias/202503/pib-fecha-2024-em-3-4-e-registra-maior-crescimento-desde-2021. Acesso em: abr. 2026.

Fundação Getúlio Vargas. **Gig Economy: o que esperar dessa tendência no Brasil**. 2025. https://www.dgabc.com.br/Noticia/4206082/gig-economy-o-que-esperar-dessa-tendencia-no-brasil. Acesso em: abr. 2026.

GetNinjas. **GetNinjas**. 2026. https://www.getninjas.com.br/. Acesso em: 20 de maio 2026.

GitHub. **GitHub Documentation**. 2026. https://docs.github.com/pt. Acesso em: 20 de maio 2026.

Google Antigravity. **Google Antigravity**. 2026. https://antigravity.google/. Acesso em: 20 de maio 2026.

Instituto de Pesquisa Econômica Aplicada. **Quantas pessoas trabalham na gig economy no Brasil?** 2025. https://startups.com.br/negocios/quantas-pessoas-trabalham-na-gig-economy-no-brasil. Acesso em: abr. 2026.

LOPES, T. R. **Método de migração de sistemas monolíticos legados para microsserviços**. 2021. Dissertação (Mestrado em Informática) – Universidade de Brasília, Brasília. Acesso em: abr. 2026. Disponível em: https://repositorio.unb.br/bitstream/10482/41178/1/2021_TaylorRodriguesLopes.pdf.

LUZ, T. M. H. d. **Sistema de agendamento de serviços: uma proposta de aplicação web**. 2024. Trabalho de Conclusão de Curso (Tecnologia em Sistemas para Internet) – Universidade Tecnológica Federal do Paraná, Guarapuava, 2022. Disponível em: https://tcc.tsi.pro.br/uploads/academic_activity/pdf/153/GP_COINT_2022_2_TAIS_MICHELE_HRYSSAI_DA_LUZ_PROJETO.pdf. Acesso em: abr. 2026.

Mercado Livre. **Mercado Livre**. 2026. https://www.mercadolivre.com.br/. Acesso em: 20 de maio 2026.

Pega Empreita. **Pega Empreita**. 2026. https://pegaempreita.com.br/. Acesso em: 20 de maio 2026.

PostgresSql. **PostgresSql Documentation**. 2026. https://www.postgresql.org/docs/. Acesso em: 20 de maio 2026.

Shopee. **Shopee**. 2026. https://shopee.com.br/. Acesso em: 20 de maio 2026.

Sociedade Brasileira de Computação. **Modernização de Arquitetura de Sistemas: uma comparativa entre os modelos monolíticos e de microsserviços**. 2024.

https://sol.sbc.org.br/index.php/eries/article/download/38408/38182. Escola Regional de Informática. Acesso em: abr. 2026.

Spartan UI. **Spartan UI Documentation**. 2026. https://www.spartan.ng. Acesso em: 18 maio 2026.

Spring Team. **Spring Framework Documentation**. 2026. https://docs.spring.io/spring-framework/reference/index.html. Acesso em: 20 maio 2026.