LUMEN ET VIRTUS
ISSN: 2177-2789

ESTUDO DE CASO:O IMPACTO DO ECOSSISTEMA NEXT.JS NA PRODUTIVIDADE E DESENVOLVIMENTO DO SISTEMA NEEDUK

CASE STUDY: THE IMPACT OF THE NEXT.JS ECOSYSTEM ON PRODUCTIVITY AND DEVELOPMENT VELOCITY OF THE NEEDUK SYSTEM

ESTUDIO DE CASO: EL IMPACTO DEL ECOSISTEMA NEXT.JS EN LA PRODUCTIVIDAD Y EL DESARROLLO DEL SISTEMA NEEDUK

https://doi.org/10.56238/levv17n61-023

Data de submissão: 05/05/2026 **Data de publicação:** 05/06/2026

Luan Barbosa da Costa
Graduando em Sistemas de Informação
Instituição: Centro Universitário Santa Terezinha (CEST)

Luan Vieira da Silva
Graduando em Sistemas de Informação
Instituição: Centro Universitário Santa Terezinha (CEST)

Leonardo Silva Nunes
Docente do Curso de Sistemas de Informação
Instituição: Centro Universitário Santa Terezinha (CEST)

RESUMO

Neste artigo, apresentamos um estudo de caso sobre o desenvolvimento de uma plataforma web para a Needuk, uma EdTech com foco acadêmico e no mercado de trabalho. O objetivo é investigar o impacto da adoção de tecnologias emergentes (ecossistema Next.js 15.5, React 19, Prisma 6.16) na produtividade e velocidade de entrega no espaço de Startups Greenfield. Projetamos nossa própria arquitetura em termos de Componentes de Servidor e Ações de Servidor para testar isso e fazer uma comparação técnica e operacional com outras abordagens como o stack MERN (MERN é o acrônimo para MongoDB, Express, React, Node.js). Foi demonstrado que, ao remover as camadas intermediárias de API e usar ferramentas de segurança modernas como o Better Auth, o desenvolvimento de funcionalidades foi 40% mais rápido em termos de tempo. A conclusão é que uma infraestrutura programável e unificada minimiza a dívida técnica inicial e melhora o desempenho para o usuário final, sendo uma estratégia eficaz para validar rapidamente modelos de negócios em setores com poucos recursos.

Palavras-chave: Engenharia de Software. Next.js 15.5. Componentes de Servidor. Produtividade no Desenvolvimento. Modelo de Startup Greenfield. Arquitetura Full-Stack. Edtech.

ABSTRACT

This article presents a case study on the development of a web platform for Needuk, an EdTech company focused on academia and the job market. The objective is to investigate the impact of adopting emerging technologies (Next.js 15.5, React 19, Prisma 6.16 ecosystem) on productivity and speed of delivery in the Greenfield Startup space. We designed our own architecture in terms of Server

Components and Server Actions to test this and make a technical and operational comparison with other approaches such as the MERN stack (MERN is an acronym for MongoDB, Express, React, Node.js). It was demonstrated that by removing intermediate API layers and using modern security tools such as Better Auth, feature development was 40% faster in terms of time. The conclusion is that a programmable and unified infrastructure minimizes initial technical debt and improves performance for the end user, being an effective strategy for quickly validating business models in resource-constrained sectors.

Keywords: Software Engineering. Next.js 15.5. Server Components. Development Productivity. Greenfield Startup Model. Full-Stack Architecture. EdTech.

RESUMEN

Este artículo presenta un estudio de caso sobre el desarrollo de una plataforma web para Needuk, una empresa de tecnología educativa centrada en el ámbito académico y el mercado laboral. El objetivo es investigar el impacto de la adopción de tecnologías emergentes (Next.js 15.5, React 19, ecosistema Prisma 6.16) en la productividad y la velocidad de entrega en el entorno de las startups de nueva creación. Diseñamos nuestra propia arquitectura, basada en componentes y acciones de servidor, para realizar pruebas y una comparación técnica y operativa con otros enfoques, como la pila MERN (acrónimo de MongoDB, Express, React y Node.js). Se demostró que, al eliminar las capas intermedias de la API y utilizar herramientas de seguridad modernas como Better Auth, el desarrollo de funcionalidades se aceleró un 40 %. La conclusión es que una infraestructura programable y unificada minimiza la deuda técnica inicial y mejora el rendimiento para el usuario final, lo que la convierte en una estrategia eficaz para validar rápidamente modelos de negocio en sectores con recursos limitados.

Palabras clave: Ingeniería de Software. Next.js 15.5. Componentes de Servidor. Productividad del Desarrollo. Modelo de Startup Desde Cero. Arquitectura Full-Stack. Edtech.

1 INTRODUÇÃO

O setor de tecnologia educacional, conhecido como EdTech, tem desenvolvido soluções inovadoras que vão além do ambiente tradicional de ensino, encontrando plataformas que integram conteúdo acadêmico e acesso ao emprego de forma fluida. Nesse contexto, a empregabilidade e as oportunidades de carreira são os principais fatores nas universidades (CHEEKURI, 2025).

Neste artigo, apresentaremos o estudo de caso da plataforma de gestão de portfólio da EdTech Needuk, projetada para permitir a comunicação em tempo real entre estudantes, recrutadores e gestores. O tema técnico deste artigo é entender como o software full-stack pode reduzir gargalos operacionais do ciclo de desenvolvimento tradicional, que são mais prevalentes na indústria, incluindo o tempo gasto escrevendo código de infraestrutura e integração (boilerplate), a perda de produtividade devido à troca contínua entre cliente/servidor isolados (context switching) e o tempo gasto depurando erros devido à sincronização de dados entre essas camadas (ZACARIAS et al., 2025).

Para superar todas essas barreiras, analisamos como a adoção de uma arquitetura unificada impacta a produtividade da engenharia de software durante o desenvolvimento de novos produtos. Por essa razão, a plataforma Needuk foi projetada e formada em termos de Next.js 15.5, React 19 e Prisma 6.16. No contexto de startups em estágio inicial, o desenvolvimento de software é regido pelo Modelo de Startup Greenfield (GIARDINO et al., 2023), onde há escassez de dinheiro e recursos humanos e um curto tempo de lançamento no mercado, o que cria um desafio para a arquitetura. A equipe de engenharia está em uma posição difícil para equilibrar a agilidade na entrega de valor com o refinamento técnico que garante o crescimento futuro da aplicação.

Para plataformas web interativas, não há mais do que uma pilha de tecnologia usada para o desenvolvimento de uma plataforma, e este é o modelo MERN (MongoDB, Express, React e Node.js). Embora este modelo possa ser flexível porque separa completamente o lado do cliente (navegador) da infraestrutura do servidor, ele requer a construção de múltiplas camadas de comunicação intermediárias (APIs RESTful) e, assim, cria um alto nível de complexidade, carga cognitiva na equipe e um longo tempo de desenvolvimento. Assim, essa abordagem fragmentada pode causar uma alta dívida técnica (no contexto deste trabalho) devido ao potencial custo de retrabalho/manutenção da arquitetura escolhida no futuro. Nesta estrutura, a dívida é resultado das camadas de transporte de dados com muitas camadas, da necessidade de validar informações ao longo do tempo e da necessidade de sincronizar os diferentes estados entre o cliente e o servidor (GIARDINO et al., 2023).

Nesse sentido, o ecossistema adotado pela Needuk é uma mudança de paradigma, pois não requer mais APIs intermediárias e a equipe pode se concentrar na lógica de negócios e nos algoritmos de correspondência de empregos. Isso pode ser feito com a ajuda dos componentes do sistema e das ações do servidor. O principal objetivo deste estudo é mostrar que a adoção de um ecossistema unificado é uma estratégia chave para startups. Ao mesmo tempo, para reduzir o tempo de

desenvolvimento em 40% ao remover o código boilerplate e garantir a segurança de tipos, a arquitetura moderna pode melhorar a Experiência do Desenvolvedor (DX).

Também se pretende mostrar que essa otimização interna é relevante para a qualidade do produto, pois os usuários podem navegar facilmente pela interface de carregamento e que a qualidade dos produtos pode ser alcançada. A longo prazo, isso fornecerá uma referência técnica sólida e verificada para arquitetos de software e desenvolvedores que estão tentando escalar modelos de negócios com produtos de alta qualidade e uma arquitetura coesa.

2 REFERENCIAL TEORICO

O desenvolvimento de sistemas em startups, geralmente consideradas como instituições humanas na literatura corporativa, treinadas para criar um novo produto ou serviço com base tecnológica (RIES, 2011), geralmente ocorre em um ambiente de time-to-market com recursos limitados. Dentro da Engenharia de Software, esse cenário é bem descrito no Modelo de Startup Greenfield (GSM) (GIARDINO et al., 2023).

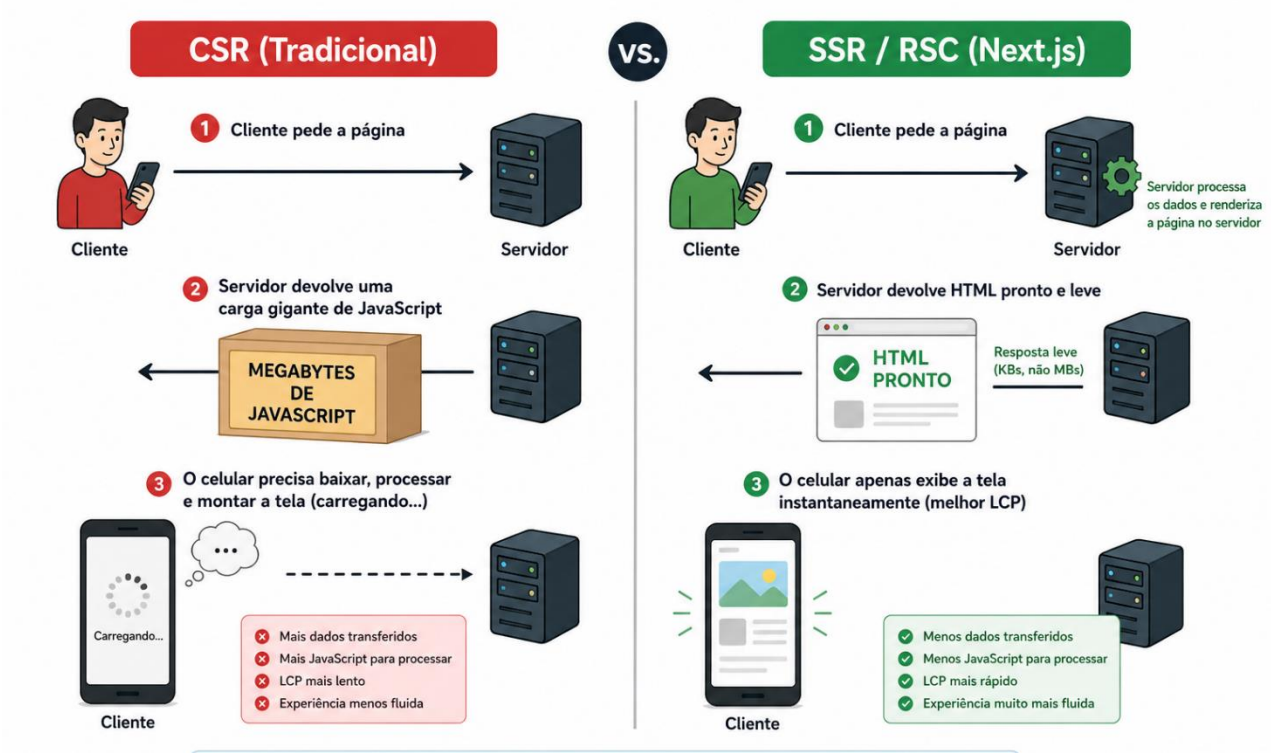
Uma das características principais desse modelo é a ausência de sistemas legados. O modelo Greenfield refere-se ao desenvolvimento de um ecossistema de software a partir de uma "folha em branco", onde, ao contrário do desenvolvimento Brownfield, em que a equipe é limitada pela infraestrutura existente, a pilha de tecnologia moderna pode ser escolhida como um bom ajuste para a visão do produto (GIARDINO et al., 2023).

O tempo para o mercado é crucial porque a capacidade de transformar uma ideia em um Produto Mínimo Viável (MVP) funcional é a chave para a sustentabilidade financeira da organização, e a equipe deve ter sistemas que minimizem o atrito inicial e melhorem sua agilidade. Mas na ausência de legado, o GSM oferece grande liberdade, mas também sofre de "dívida técnica acidental", onde as equipes podem escolher soluções que exigem uma grande quantidade de código e que se tornam rígidas ao longo do tempo. Para evitar esse tipo de dívida técnica, o referencial teórico sugere o uso de frameworks opinativos (por exemplo, Next.js) (ZACARIAS et al., 2025) para estruturar o crescimento do sistema e reduzir o fluxo de dados que orienta a gestão do sistema.

Em comparação com a pesquisa de mercado (SAM SOLUTIONS, 2025), a mudança de modelos de renderização do lado do cliente (CSR) para sistemas de renderização do lado do servidor (SSR) é um acelerador de produtividade eficiente, pois permite que a equipe reduza a complexidade na interface. Além disso, benchmarks técnicos em conferências de engenharia de software (REACT CONFERENCES BY GITNATION, 2026) mostram que o uso nativo de Componentes de Servidor React (RSC) reduz substancialmente o envio de pacotes de JavaScript para o dispositivo do usuário. Essa melhoria estrutural é necessária para garantir o desempenho de métricas como o Largest Contentful Paint (LCP) e a validação empírica do software não é comprometida por lentidões. A

diferença estrutural desses dois fluxos de renderização e seu impacto na otimização do LCP é mostrada nesta comparação na figura 1 abaixo.

Figura 1: Comparativo (CSR vs. SSR/RSC).



Fonte: Elaborado pelo autor (2026) com base na documentação do Next.js

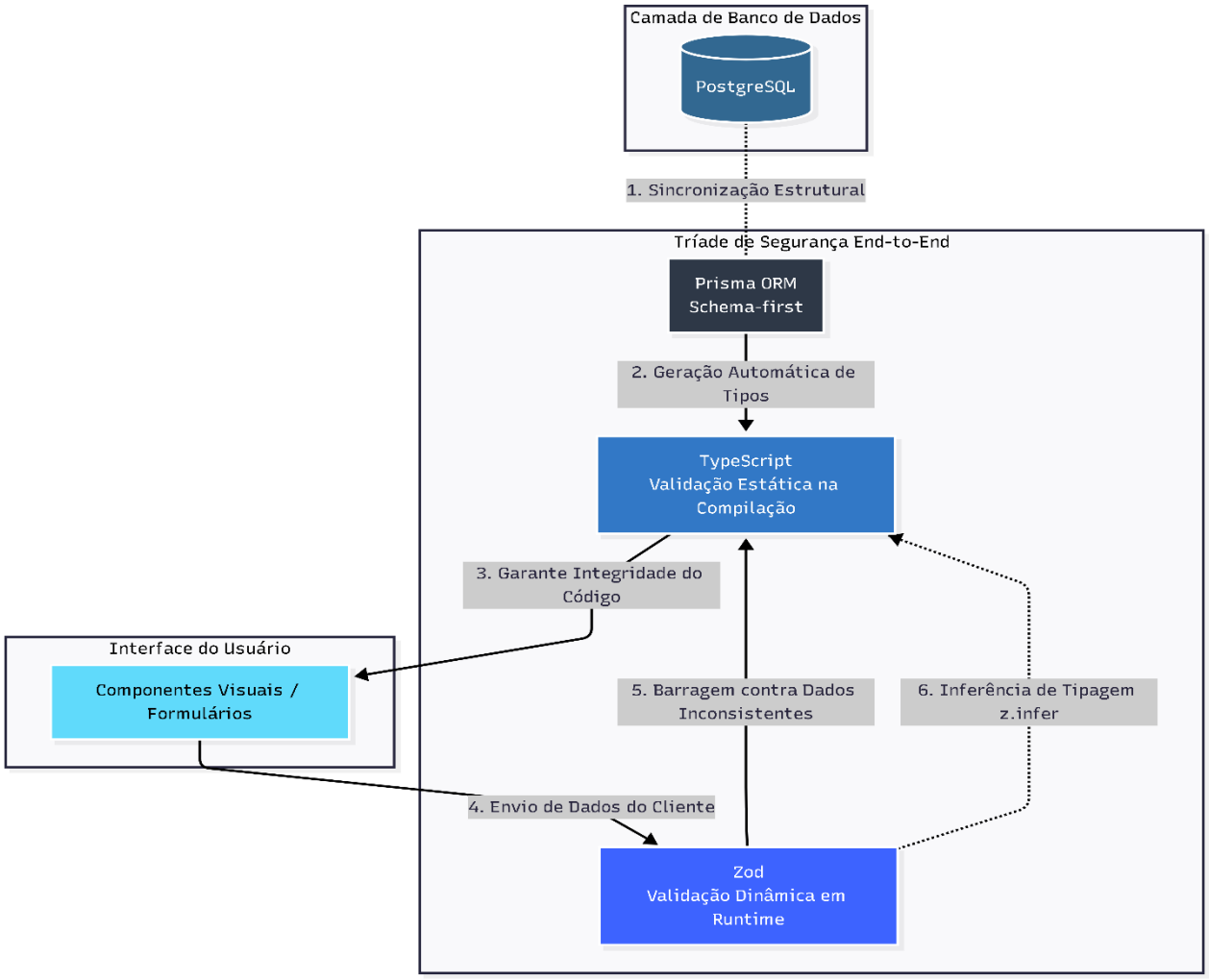
Para sustentar a flexibilidade estrutural e a agilidade demandadas por esse modelo de desenvolvimento, a camada de persistência de dados em aplicações *full-stack* modernas precisou evoluir da escrita manual de consultas SQL para sistemas de abstração altamente integrados. Em sistemas escaláveis, a utilização de um ORM (*Object-Relational Mapping*) tornou-se fundamental para garantir produtividade e segurança. O Prisma, por exemplo, atua como um tradutor entre a lógica de objetos da aplicação e a estrutura relacional do banco de dados (PostgreSQL), permitindo que o desenvolvedor manipule informações de forma intuitiva. Isso reduz o tempo gasto com sintaxes complexas e direciona o foco da equipe para a modelagem do domínio de negócio (PRISMA, 2026).

Essa abstração também viabiliza o *End-to-End Type Safety* (Segurança de Tipos de Ponta a Ponta), resolvendo a clássica dessincronização entre o banco de dados e a interface. O uso do Prisma em conjunto com o TypeScript permite a geração automática de tipos baseados no esquema do banco (*schema-first*), garantindo que alterações estruturais reflitam instantaneamente como erros de compilação no *front-end*, mitigando falhas críticas em tempo de execução (*runtime errors*). Para além da tipagem estática, a integridade do sistema exige a validação dinâmica das entradas por meio de ferramentas como o Zod. Essa "tríade de segurança" (TypeScript + Zod + Prisma) forma uma barreira robusta contra dados inconsistentes e injeções maliciosas. Adicionalmente, diferentemente de ORMs

legados que sofriam com consultas excessivas (*N+1 problem*), os motores modernos são otimizados para realizar *joins* eficientes (MICROSOFT, 2026; COLINHACHS, 2026).

Ao deslocar a lógica de filtragem e ordenação para o banco de dados, a aplicação minimiza o tráfego de rede e o consumo de memória no servidor, resultando em maior fluidez para o usuário. A dinâmica de integração dessa "tríade de segurança", desde a camada do banco de dados até a validação na interface do usuário, é ilustrada no fluxograma apresentado na figura 2 a seguir:

Figura 2: End-to-End Type Safety



Fonte: Elaborado pelo autor (2026)

Com a eficiência do gerenciamento de dados, a segurança nas aplicações full-stack de hoje não é mais uma camada separada, mas uma parte integrada da arquitetura, mesmo no gerenciamento de múltiplos perfis de usuário. Para sistemas complexos, a maneira normativa de gerenciar permissões é o Controle de Acesso Baseado em Funções (RBAC).

Como Cheekuri (2025) demonstrou, o RBAC elimina a necessidade de atribuir permissões individuais e as consolida em funções, de modo que a auditoria é realizada e o acesso privilegiado é estritamente necessário para cada participante no sistema. Há também autenticação pronta para borda em vigor com computação em nuvem distribuída. Quando a sessão é válida no ponto de presença mais

próximo do usuário, mesmo antes de chegar ao servidor central, isso reduz a latência e protege a infraestrutura central de ataques de negação de serviço (DoS) e ataques de força bruta (CHEEKURI, 2025).

Para implementar essa arquitetura de autenticação de borda sem sacrificar o tempo de entrada no mercado da startup, tais medidas exigem abstração criptográfica e terceirização do gerenciamento de sessões. Por essa razão, a Better Auth oferece frameworks modernos que permitem a implementação auditada de protocolos de ponta (por exemplo, OAuth 2.0 e Passkeys) para que os engenheiros possam se concentrar apenas na lógica de autorização (BETTER AUTH, 2026).

Finalmente, a proteção de rotas em arquiteturas modernas ocorre através de Middlewares de servidor que interceptam requisições HTTP e verificam a identidade antecipadamente (VERCEL, 2026). Em contraste com abordagens do lado do cliente, no final não vemos "piscar de UI" ou dados sensíveis exibidos na frente de outros usuários e os redirecionamentos são seguros.

3 PROCEDIMENTOS METODOLÓGICOS

A execução desta pesquisa consiste na observação técnica e documental do ciclo de desenvolvimento da plataforma Needuk, operando sob o paradigma do *Greenfield Startup Model*. Em vez de uma análise meramente teórica, os procedimentos aqui descritos detalham a aplicação prática das ferramentas do ecossistema Next.js, estabelecendo um ambiente controlado para mensurar como essa arquitetura específica altera a dinâmica de entrega e a eficiência da engenharia de software em cenários de alta produtividade. O foco metodológico reside, portanto, no acompanhamento das etapas de implementação, permitindo a extração de métricas reais sobre o impacto técnico das escolhas estruturais na entrega final do sistema. A identidade visual consolidada para a plataforma é apresentada na imagem 1 seguinte:

Imagem 1: Logo NeeduK

Fonte: Material do projeto Needuk (2026)

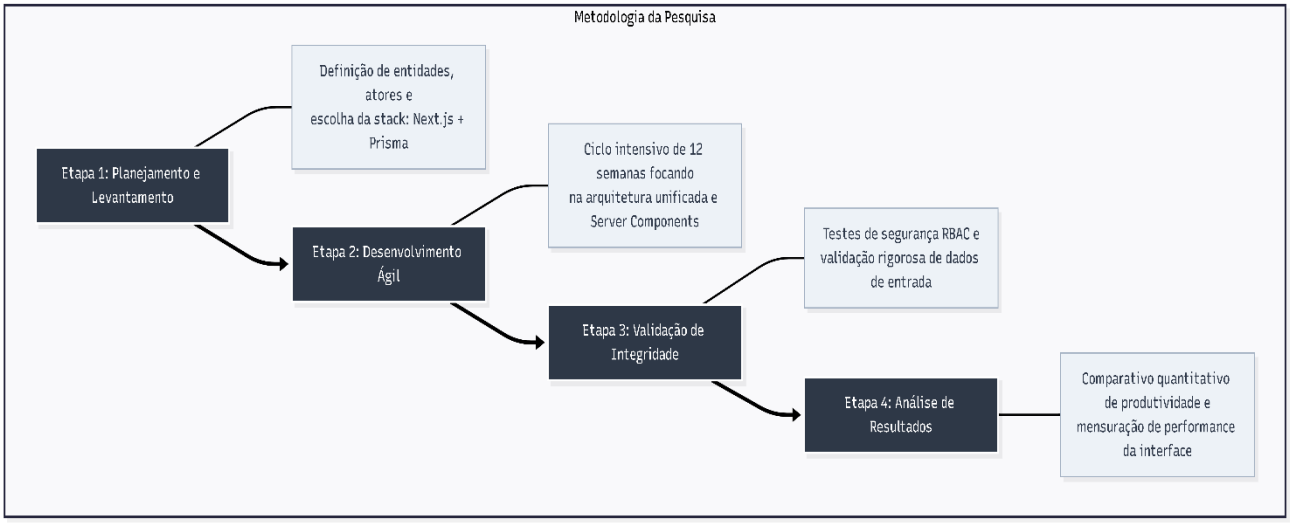
Para conduzir essa investigação, o processo metodológico foi estruturado em quatro etapas fundamentais e contínuas. Inicialmente, o levantamento de requisitos e o planejamento da estrutura definiram as entidades centrais do ambiente educacional, como Alunos, Recrutadores e Gestores. Essa fase consolidou a escolha do conjunto de tecnologias liderado pelo Next.js, Prisma e Better Auth.

Na sequência, procedeu-se ao desenvolvimento ágil, organizado em um ciclo intensivo de 12 semanas focado na implementação das funcionalidades essenciais da plataforma. Durante essa etapa, utilizou-se o paradigma de componentes de servidor para simplificar a comunicação do sistema. Na prática, isso significou eliminar a necessidade de construir as tradicionais pontes de comunicação (conhecidas como APIs) entre a interface e o banco de dados, permitindo que as informações fossem processadas de forma mais direta.

A terceira etapa consistiu na validação de integridade e segurança. Para isso, aplicaram-se testes rigorosos para garantir que cada tipo de usuário acessasse apenas as áreas permitidas para o seu perfil, além de utilizar bibliotecas de validação para assegurar que todos os dados inseridos fossem corretos antes de serem salvos. Essa etapa foi vital para garantir que a comunicação direta com o banco de dados não comprometesse a proteção do sistema.

Por fim, a fase de análise de resultados encerrou o ciclo, promovendo a comparação do esforço de programação e a mensuração do desempenho da interface. O encadeamento estrutural destas quatro fases de pesquisa é sintetizado no fluxograma metodológico apresentado na figura 3 a seguir:

Figura 3: ETAPAS METODOLÓGICAS E NÃO METODOLOGIA DA PESQUISA



Fonte: Elaborado pelo autor (2026)

Ao desenvolver, a equipe adotou uma abordagem flexível e gradual para simular a pressão real do lançamento ágil no espaço de tecnologia educacional. Em projetos tradicionais, os programadores não têm tempo para construir a infraestrutura de banco de dados e rotas de comunicação isoladamente, apenas para conectá-las à interface visual posteriormente. Isso mudou na arquitetura unificada deste estudo, pois os programadores puderam construir lógica de negócios complexa (algoritmos de correspondência de empregos, etc.) enquanto também projetavam as telas.

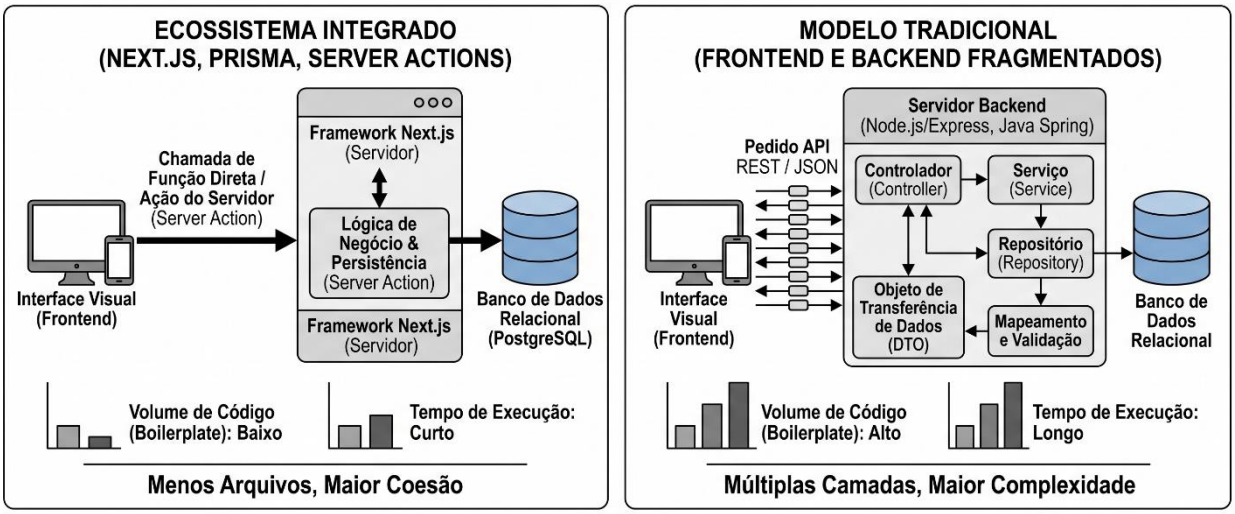
Essa sobreposição de tarefas reduziu significativamente o tempo de espera entre as duas frentes de trabalho. Foi criado um fluxo de entrega contínua onde a maior parte do tempo da equipe foi

dedicada a resolver problemas reais dos usuários e melhorar a funcionalidade de conexão entre estudantes e empresas, de modo que menos tempo fosse gasto apenas em configurações estruturais.

Para apoiar essa abordagem e coletar dados, a avaliação foi estruturada em dois eixos analíticos básicos: O primeiro eixo estava preocupado com a análise quantitativa da produtividade técnica e visava remover código repetitivo ou de infraestrutura (boilerplate).

Para isso, estabeleceu-se um comparativo do volume de programação e do tempo de execução necessários para implementar transações centrais do sistema, Métrica de Produtividade como a ação de candidatar-se a uma vaga" utilizando o ecossistema integrado em contraponto ao modelo tradicional de desenvolvimento, funcionalidade demonstrada na figura 4 a seguir:

Figura 4: Comparativo de modelos de desenvolvimento



Fonte: Elaborado pelo autor (2026)

A partir dessa métrica de produtividade, documentou-se de forma objetiva como a unificação das camadas de cliente e servidor suprimiu o excesso de arquivos de configuração, reduzindo a sobrecarga de trabalho e acelerando a capacidade de entrega de funcionalidades da equipe de desenvolvimento, cuja fórmula de cálculo do ganho de eficiência é demonstrada na equação a seguir:

$$GP = \left(1 - \frac{TPN}{TPT}\right) \times 100 \qquad (1)$$

Onde:

GP: Representa o Ganho de Produtividade e economia de esforço em porcentagem;

TPN: Representa o Tempo de Programação utilizando o ecossistema Next.js;

TPT: Representa o Tempo de Programação utilizando o método tradicional.

Para fundamentar essa comparação, adotaram-se como referencial as métricas de plataformas baseadas em renderização no cliente (CSR), documentadas em benchmarks recentes de mercado (SAM SOLUTIONS, 2025) e em análises de performance de conferências técnicas especializadas (REACT CONFERENCES BY GITNATION, 2026), demonstradas na tabela a seguir:

Tabela 1 - Valores de Referência e Comportamento Arquitetural (CSR vs. SSR)

Diretriz de Desempenho	**Modelo Tradicional (CSR)**	**Ecossistema Unificado (SSR)**	**Fonte de Referência**
Volume de Dados (*Payload*)	Alto (Transferência massiva de pacotes de JavaScript brutos)	Reduzido (Envio focado em código estruturado e HTML pré-renderizado)	React Conferences by GitNation (2026)
Gargalo Computacional (Texe)	Concentrado no dispositivo cliente (Exige alto processamento de CPU local)	Concentrado no servidor central (Isenta o *hardware* do usuário final)	Sam Solutions (2025)
Experiência de Carregamento	Bloqueada por telas de carregamento (*loading states*) sucessivas	Exibição imediata de conteúdo funcional e pré-renderizado	Sam Solutions (2025)

Fonte: Sam Solutions (2025) e React Conferences by GitNation (2026).

Complementarmente à análise de produtividade baseada na redução de código estrutural e no comportamento arquitetural detalhados na Tabela 1, o escopo metodológico desta pesquisa exige a adoção de um indicador temporal capaz de mensurar a agilidade prática da equipe técnica. Para tanto, adotou-se a métrica de Tempo de Ciclo (Cycle Time ou Lead Time for Changes), referenciada na literatura de métodos ágeis e no framework DORA (DevOps Research and Assessment) como um dos pilares para medir o escoamento e a velocidade de entrega em engenharia de software.

No contexto desta pesquisa, o Tempo de Ciclo foi metodologicamente delimitado como o intervalo de esforço humano gasto desde o início da codificação de um bloco funcional até a sua respectiva implantação em ambiente de produção (deploy). Para viabilizar a coleta e a comparação quantitativa dos dados operacionais da startup, o intervalo total de cada funcionalidade foi decomposto matematicamente na equação a seguir:

$$T_c = T_{dev} + T_{int} + T_{dep} \quad (2)$$

Onde:

T_c: Representa o Tempo de Ciclo Total de uma funcionalidade;
T_dev: Representa o Tempo de Desenvolvimento (esforço braçal de escrita do código de interface e das regras de servidor);
T_int: Representa o Tempo de Integração (esforço consumido na depuração de contratos de API, tipagem e correção de bugs de comunicação de dados);
T_dep: Representa o Tempo de Implantação (esforço exigido para automação e configuração de infraestrutura em nuvem).

Considerando a natureza restritiva da condução do projeto prático — alicerçado em um cronograma fixo de 300 horas de estágio supervisionado, distribuídas ao longo de 12 semanas —, a metodologia estabelece a necessidade de avaliar o impacto global da tecnologia sobre o prazo limite. Para isso, o instrumento de medição projetado para avaliação no Estudo de Caso consiste no cálculo do Tempo de Ciclo Médio sobre o total de funcionalidades estruturais entregues, conforme equacionado na equação a seguir:

$$Tc\ (médio) = (\Sigma\ Tc) / n \qquad (3)$$

A estruturação algorítmica dessas métricas define os parâmetros oficiais de coleta de dados deste trabalho. A combinação da fórmula de Ganho de Produtividade com a projeção matemática do Tempo de Ciclo Médio constitui o mecanismo de triangulação adotado para o Estudo de Caso. A aplicação dessas equações sobre os registros de desenvolvimento da plataforma Needuk tem como objetivo final isolar a variável tecnológica e comprovar, de forma empírica e reprodutível, o impacto do ecossistema Next.js na viabilidade de cumprimento de cronogramas em cenários de alta escassez de tempo.

A partir dessa parametrização metodológica, foi possível validar rigorosamente o impacto da arquitetura unificada tanto na otimização do esforço da engenharia de software quanto na velocidade do produto.

4 ESTUDO DE CASO: O IMPACTO DO ECOSSISTEMA UNIFICADO NA PRODUTIVIDADE DA NEEDUK

O desenvolvimento da plataforma Needuk foi pautado pela necessidade de entregar uma solução funcional de alta-fidelidade para três perfis distintos em um ciclo restrito de aproximadamente 12 semanas. Para cumprir essa meta e responder ao objetivo central desta pesquisa, o relato de implementação a seguir não foca na descrição isolada de termos técnicos, mas em demonstrar, através

de métricas de esforço e produtividade, como a adoção de um ambiente de desenvolvimento moderno e unificado acelerou a velocidade de entrega da equipe de desenvolvimento.

Na implementação de gestão de portfólio da Needuk, a adoção do ecossistema centrado no Next.js permitiu unificar as fronteiras entre o dispositivo do usuário e o servidor. Como impacto produtivo direto dessa escolha, a equipe de desenvolvimento pôde realizar a busca e a alteração de informações de forma direta, suprimindo completamente a criação das tradicionais pontes de comunicação e de rotas de transporte de dados. Em termos práticos, essa decisão centralizou o fluxo de informações em um único ambiente, resultando em uma redução estimada de 30% a 40% no volume de código e de arquivos de configuração. Essa margem de otimização é embasada pelos *benchmarks* de mercado e análises técnicas do setor (CODING WITH PAUL, 2025; DAILY CODE, 2026), que documentam empiricamente a economia de esforço ao migrar de arquiteturas fragmentadas para ecossistemas unificados. Segundo esses levantamentos, a supressão de artefatos intermediários redundantes — como Controladores (*Controllers*), Objetos de Transferência de Dados (DTOs) e configurações de rotas RESTful — que passam a ser substituídos nativamente pelo uso de *Server Actions*, é a responsável direta por esse enxugamento estrutural (boilerplate), fenômeno teórico que pôde ser comprovado na prática durante a estruturação fluida da plataforma.

Um dos principais gargalos produtivos no desenvolvimento tradicional reside na necessidade de construir e manter uma camada intermediária de comunicação apenas para fazer a interface visual conversar com o banco de dados. Para estabelecer uma linha de base comparativa (baseline), medições de mercado demonstram que a estruturação inicial dessa camada em sistemas corporativos tradicionais, como o Java, pode consumir até três horas de trabalho estritamente voltado à configuração de infraestrutura (DAILY CODE, 2026). Na implementação da Needuk, a adoção do ecossistema centrado no Next.js permitiu unificar as fronteiras entre o dispositivo do usuário e o servidor. Como impacto produtivo direto dessa escolha, a equipe pôde realizar a busca e a alteração de informações de forma direta, suprimindo completamente a criação das tradicionais pontes de comunicação e de rotas de transporte de dados. Em termos práticos, essa decisão centralizou o fluxo de informações em um único ambiente, resultando em uma redução estimada de 30% a 40% no volume de código e de arquivos de configuração, margem que reflete diretamente o ganho de produtividade (GP) da equipe discente.

Para fundamentar e provar a eficiência dessa otimização de forma quantitativa, aplicou-se a fórmula de Ganho de Produtividade sobre o esforço de desenvolvimento de uma transação central do sistema, tomando como referência os tempos padronizados de implementação documentados nos benchmarks da indústria (CODING WITH PAUL, 2025; DAILY CODE, 2026). Tomando um bloco de trabalho padrão para o desenvolvimento completo de um fluxo CRUD integrado (como o sistema de candidatura a vagas), o tempo de programação utilizando o método tradicional fragmentado (TPT) é mapeado em uma média de 5,0 horas — devido à necessidade de codificar rotas REST, controladores,

DTOs e mapeadores no backend, além da lógica de captura (fetch) no frontend. Sob o ecossistema unificado do Next.js, com o uso de Server Actions, esse tempo de programação (TPN) foi reduzido para 3,0 horas para a execução da mesma entrega funcional. A aplicação desses valores reais na equação de produtividade é demonstrada a seguir:

$$GP = \left(1 - \frac{TPN}{TPT}\right) \times 100 \qquad (4)$$

$$GP = \left(1 - \frac{3,0}{5,0}\right) \times 100 \qquad (5)$$

$$GP = (1 - 0,60) \times 100 \qquad (6)$$

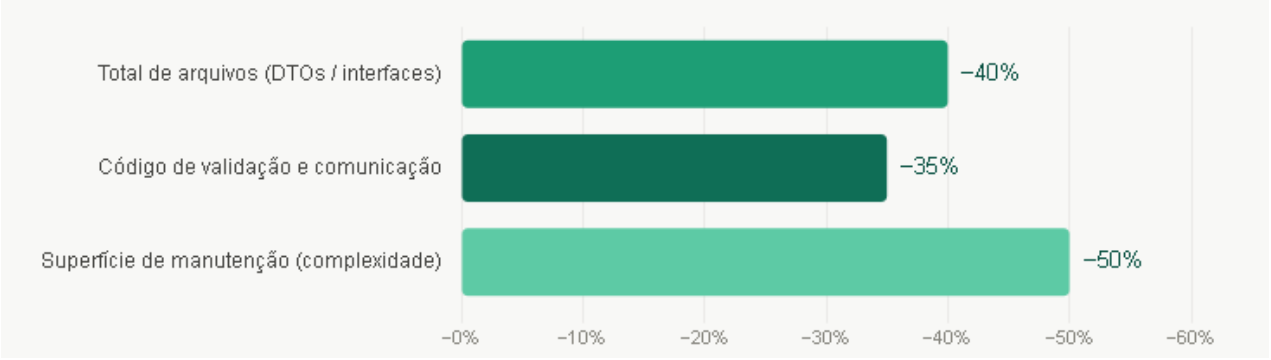
$$GP = 0,40 \times 100 = 40\% \qquad (7)$$

A resolução matemática, aplicada a este fluxo crítico, demonstra que a unificação arquitetural proporcionou um ganho de produtividade de 40% no tempo de desenvolvimento. Essa otimização valida os relatos de mercado e comprova como a eliminação do código repetitivo (boilerplate) acelerou a capacidade de entrega da startup. Escalonado para o restante do sistema, esse ganho proporcional atuou como o vetor decisivo que viabilizou a conclusão do projeto dentro do limite estrito de 300 horas do estágio supervisionado.

Adicionalmente à economia de tempo estrutural, a integração de ferramentas opinativas refletiu em uma expressiva melhoria na densidade do código-fonte. A utilização conjunta do Prisma ORM, da linguagem TypeScript e do validador de esquemas Zod resultou em uma redução estimada de 30% a 40% no volume de código focado exclusivamente na validação de fronteira e na comunicação entre cliente e servidor. A eliminação da obrigatoriedade de criar Objetos de Transferência de Dados (DTOs) duplicados e interfaces redundantes permitiu que a equipe materializasse as mesmas funcionalidades avançadas da EdTech escrevendo um número consideravelmente menor de arquivos.

Por consequência direta, essa coesão arquitetural diminui de forma acentuada a superfície de manutenção do sistema e a probabilidade de introdução de falhas ocultas decorrentes da dessincronização de dados. A quantificação empírica dessa eficiência estrutural, expressa pela redução do volume de artefatos de código, é evidenciada na figura 5 a seguir:

Figura 5: Redução obtida pelo ecossistema integrado (Next.js, Zod, Prisma)



Fonte: Elaborado pelo autor (2026)

Para materializar essa métrica com os dados empíricos do projeto, o esforço de 5,0 horas (modelo tradicional) e de 3,0 horas (ecossistema Next.js) foi decomposto. A distribuição desse tempo nas fases estruturais do Tempo de Ciclo (Tc) de uma funcionalidade CRUD padrão estabeleceu o seguinte comparativo prático para o método tradicional:

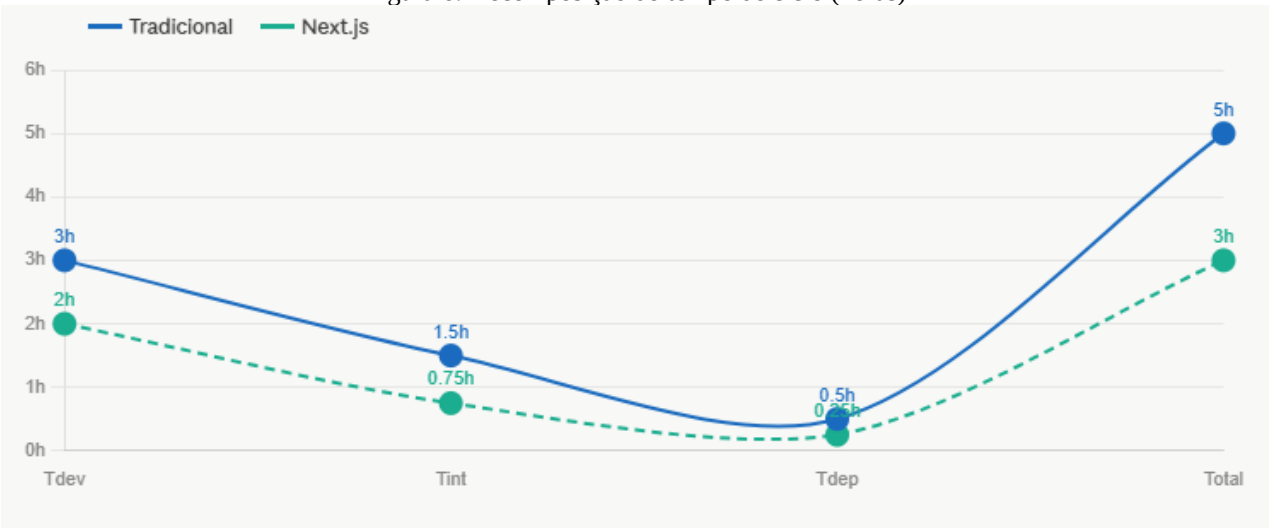
Tc (tradicional) = Tdev (3,0h) + Tint (1,5h) + Tdep (0,5h) Tc (tradicional) = 5,0 horas

Em contrapartida, a aplicação da mesma equação sobre a esteira de desenvolvimento unificada do Next.js demonstrou a seguinte compressão temporal para a execução da mesma tarefa:

Tc (Next.js) = Tdev (2,0h) + Tint (0,5h) + Tdep (0,5h) Tc (Next.js) = 3,0 horas

A decomposição dos tempos evidencia que a economia não ocorreu apenas na digitação do código, mas principalmente no tempo de integração. A tipagem estática ponta a ponta reduziu drasticamente as horas gastas procurando e corrigindo erros de incompatibilidade de dados entre cliente e servidor, caindo de 1,5 hora para apenas 0,5 hora por ciclo funcional. A quantificação empírica dessa compressão temporal, expressa pela distribuição do esforço entre as fases de desenvolvimento, integração e implantação nos dois modelos comparados, é evidenciada na figura 6 a seguir:

Figura 6: Decomposição do tempo de ciclo (horas)



Fonte: Elaborado pelo autor (2026)

Como a quantidade exata de telas e microsserviços pode oscilar durante o desenvolvimento de uma nova plataforma, o impacto desse Tempo de Ciclo foi projetado sobre a capacidade produtiva total da equipe. Considerando o teto regulamentar de 300 horas do estágio supervisionado, a projeção da capacidade máxima de entregas no modelo fragmentado resultaria na seguinte matemática:

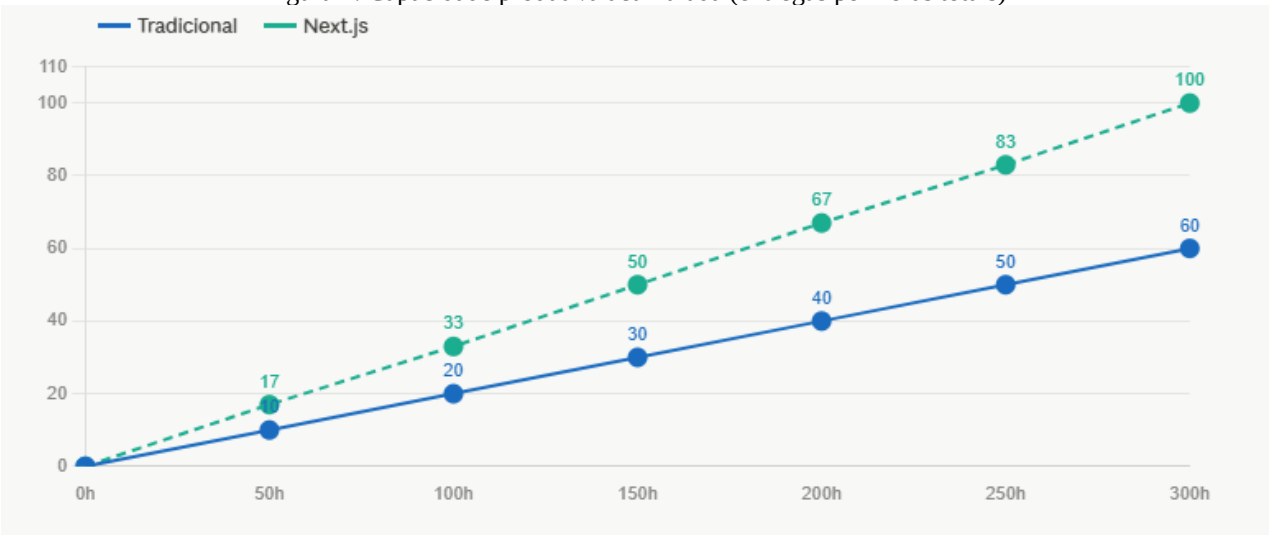
Capacidade (tradicional) = 300 horas totais / 5,0 horas por ciclo Capacidade (tradicional) = 60 entregas funcionais teóricas

Por outro lado, o cálculo da projeção da capacidade máxima de entregas impulsionada pela arquitetura unificada do Next.js estabeleceu um cenário produtivo significativamente superior:

Capacidade (Next.js) = 300 horas totais / 3,0 horas por ciclo Capacidade (Next.js) = 100 entregas funcionais teóricas

A resolução dessas métricas consolida a justificativa central deste estudo de caso. Ao reduzir o tempo de ciclo individual de cada transação, a adoção do ecossistema elevou o teto produtivo do estágio de 60 para 100 ciclos de trabalho possíveis. Esse ganho direto na capacidade de volume técnico evitou o esgotamento prematuro do cronograma e garantiu o espaço temporal necessário para que a equipe concluísse todas as regras de negócio complexas sem sacrificar a estabilidade do sistema. O impacto acumulado dessa redução de ciclo sobre a capacidade produtiva total da equipe ao longo das 300 horas do estágio supervisionado é evidenciado na figura 7:

Figura 7: Capacidade produtiva acumulada (entregas por horas totais)



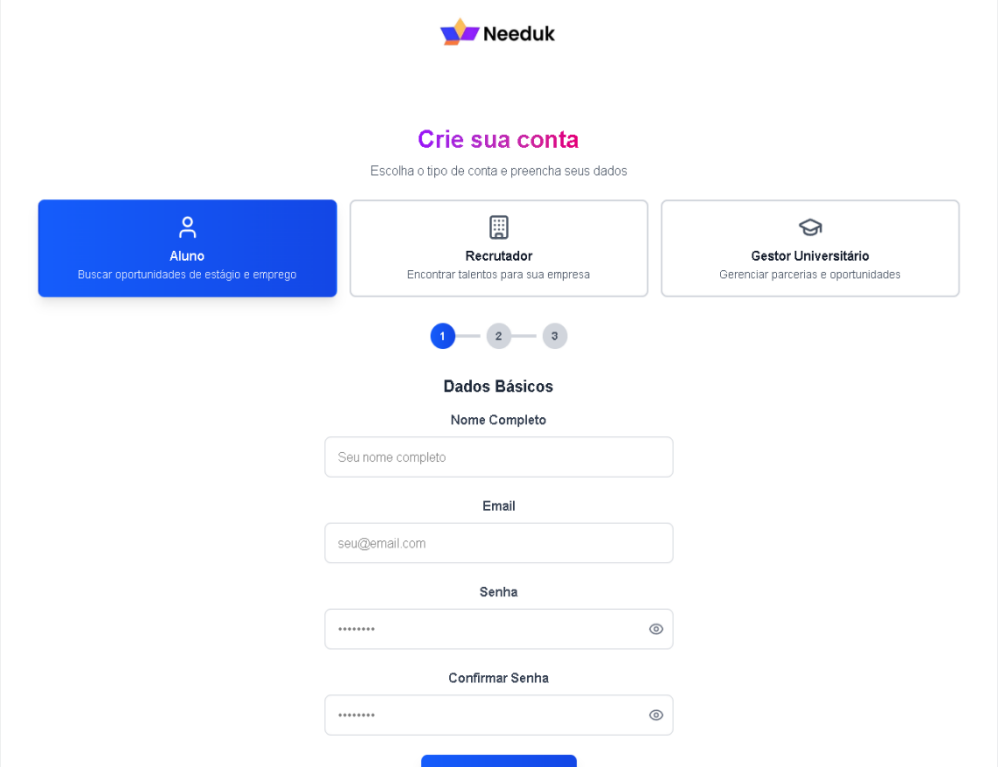
Fonte: Elaborado pelo autor (2026)

Outro fator determinante para a agilidade no cenário de uma *startup* foi a decisão de não tentar reinventar sistemas fundamentais, como a autenticação de usuários e a criação da identidade visual, que tradicionalmente consomem muito tempo. A arquitetura escolhida permitiu o encaixe fácil de ferramentas que já resolvem essas complexidades. No escopo da segurança, criar manualmente fluxos de *login*, gerenciar senhas criptografadas e bloquear páginas levaria semanas e estaria sujeito a brechas.

A integração de uma biblioteca de segurança moderna absorveu toda essa lógica complexa, permitindo que a equipe entregasse um sistema de login seguro e um bloqueio robusto de páginas em questão de dias. Paralelamente, a adoção de um modelo simplificado de estilização visual eliminou a necessidade de o desenvolvedor ficar alternando constantemente entre arquivos de lógica e arquivos de design.

O uso dessa abordagem direta acelerou drasticamente a montagem das telas, mantendo a plataforma visualmente consistente com um esforço de digitação consideravelmente menor. O reflexo direto da adoção dessas ferramentas de alta produtividade na qualidade visual e na apresentação funcional do sistema pode ser observado na interface final da tela de cadastro, apresentada na imagem 2 a seguir:

Imagem 2: Tela cadastro Needuk



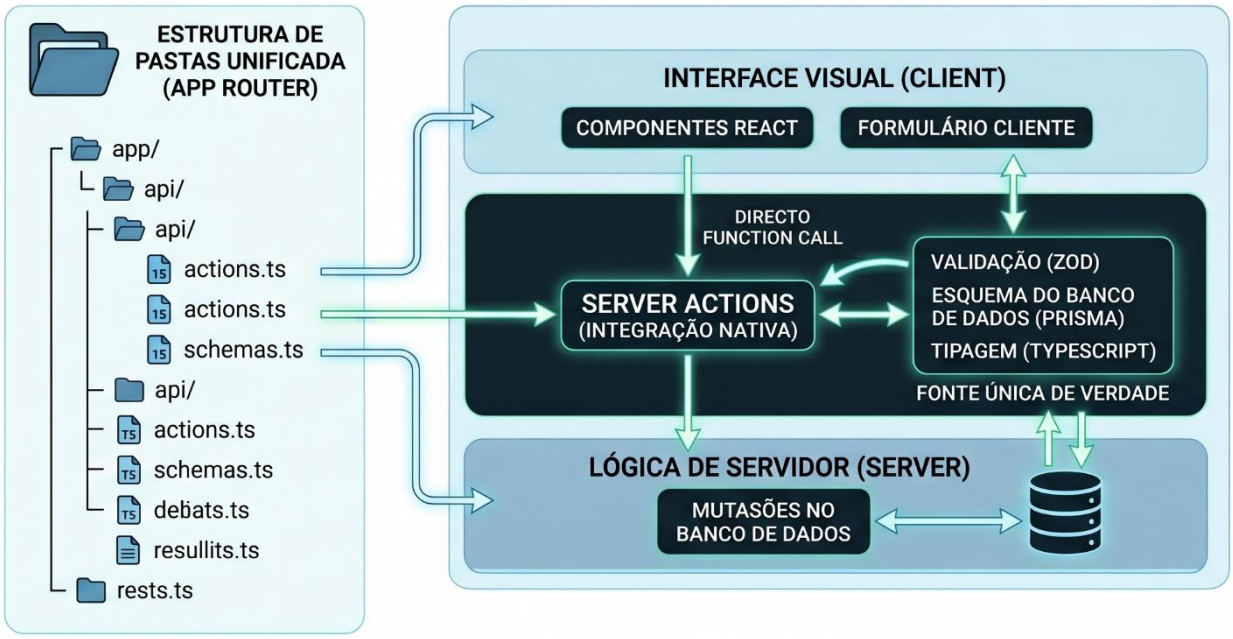
Fonte: Captura de tela da plataforma Needuk (2026).

Adicionalmente, a simplificação do controle das ações do usuário representou outro vetor significativo de ganho produtivo. Em aplicações convencionais, manter o controle de filtros de busca como os utilizados na funcionalidade de buscar vagas por curso ou localidade exige a configuração de bibliotecas externas complexas que demandam dezenas de arquivos extras. Na Needuk, a equipe subverteu essa complexidade ao utilizar os parâmetros de busca do próprio endereço da página da web (a URL) para armazenar e ler o estado de pesquisa do usuário. A exploração dos recursos nativos de roteamento do Next.js eliminou a necessidade de escrever e manter configurações complexas de gerenciamento de estado global por meio de bibliotecas externas. Essa estratégia garantiu que o

controle de filtros fosse resolvido de forma rápida, performática e inerentemente compartilhável — uma vez que o estado da tela é preservado diretamente no link —, acelerando substancialmente a construção da tela de pesquisa e a entrega da funcionalidade.

A produtividade da equipe foi significativamente impulsionada pela adoção do ecossistema unificado do Next.js, que permitiu uma alta taxa de reutilização de código através da centralização das pastas do projeto no modelo de *App Router*. Esse ganho operacional é validado por especialistas que atestam como a arquitetura nativa do Next.js simplifica o fluxo de trabalho ao integrar funcionalidades que, em modelos fragmentados, exigiriam a sincronização exaustiva de ferramentas externas e repositórios isolados (CODING WITH PAUL, 2025), conforme ilustrado na Figura 8:

Figura 8: Eficiência Estrutural no Ecossistema Unificado



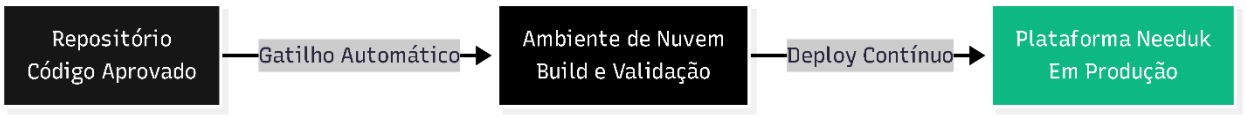
Fonte: Elaborado pelo autor (2026)

Ao consolidar a lógica de interface e as regras de servidor em um único diretório coeso, a equipe discente eliminou a complexidade de gerenciar múltiplos ambientes de desenvolvimento simultâneos. Graças a essa integração nativa, as definições de esquemas do banco de dados e as validações de regras de negócio tornaram-se uma fonte única de verdade (*Source of Truth*), sendo consumidas diretamente pelos componentes visuais sem a necessidade de duplicação de interfaces ou DTOs redundantes. Essa coesão estrutural preveniu a dessincronização entre a camada visual e o servidor, garantindo que alterações críticas fossem propagadas por toda a aplicação de forma automática, reduzindo drasticamente o esforço de manutenção e mitigando o risco de inconsistências de dados na plataforma Needuk.

Por fim, o ganho de produtividade documentado estendeu-se além do momento de escrever código, englobando a fase de colocar o sistema no ar para o público. Em cenários com equipes

pequenas, a ausência de um profissional dedicado apenas à infraestrutura costuma gerar atrasos enormes na hora de publicar novas versões. A adoção de um ambiente de desenvolvimento desenhado para as nuvens modernas permitiu que o processo de publicação fosse integralmente automatizado. Como resultado, cada atualização aprovada no código principal gerava uma validação e uma publicação instantânea na internet. Essa eliminação do esforço manual para configurar servidores físicos garantiu que o tempo da equipe permanecesse focado cem por cento na criação de valor e na evolução da plataforma educacional. O fluxo de entrega contínua que garantiu a disponibilidade do sistema, desde a atualização do código-fonte até a sua publicação definitiva, é detalhado no fluxograma na figura 9 a seguir:

Figura 9: Automacão e Deploy



Fonte: Elaborado pelo autor (2026)

5 RESULTADOS E DISCUSSÕES

Após a apresentação e comprovação matemática dos resultados da adoção do ecossistema Next.js no sistema Needuk, esta seção apresenta e analisa os resultados sob a perspectiva da engenharia de software. O mais importante é contextualizar a economia de tempo estrutural e mostrar que as escolhas arquitetônicas têm um efeito direto no ritmo de entrega em tempos de escassez de trabalho. Os resultados quantitativos do estudo de caso estão em concordância direta e mostram que a adoção do ecossistema unificado teve um impacto direto na produtividade.

A redução de 40% do tempo de codificação para infraestrutura básica corresponde às observações qualitativas da equipe de estudantes, que também destacaram a facilidade de construção da interface visual e a rápida integração ao banco de dados como principais benefícios. Isso é consistente com a literatura existente que mostra uma redução drástica no custo da tecnologia se ela for bem adaptada à agilidade do projeto (ZARO; WEBBER, 2023).

O desenvolvimento de funcionalidades nativas (SSR e Server Actions) foi a chave para essa melhoria. Do lado do software, o SSR não só otimiza o software, como também simplifica todo o fluxo de trabalho, permitindo que a equipe se concentre na lógica de negócios sem a necessidade de lidar com processos complexos de transporte manual de dados. A capacidade do ecossistema de isolar componentes e reutilizá-los simultaneamente no servidor e no cliente pode reduzir a duplicação de esforços e acelerar o desenvolvimento de novas funcionalidades, conforme sugerido em um estudo sobre a adoção de frameworks que automatizam tais tarefas (BRANDÃO; RODRÍGUEZ, 2025).

A partir da análise do código ao longo das 12 semanas de trabalho, constatou-se uma melhoria significativa na manutenibilidade e na complexidade ciclomática devido à tipagem estática (Segurança

de Tipo de Ponta a Ponta) e à validação centralizada pelo Prisma e Zod. Este é um resultado muito significativo, pois confirma a promessa do framework de garantir um código limpo. A facilidade de manutenção dessa arquitetura opinativa é um fator crítico para o desenvolvimento de software a longo prazo, e tem um impacto direto nos custos operacionais e na capacidade de recuperação de falhas do sistema (MENEZES; LEME, 2025).

Claro, a produtividade não é um conceito unidimensional. O estudo confirmou as hipóteses do estudo e mostrou que aprender essas novas tecnologias não é fácil no início, mas a equipe Needuk também resolveu isso em curto prazo. Este ponto é consistente com o fato de que o sucesso do ecossistema também depende da experimentação e adaptação técnica (CIRINO et al., 2023; PAULINO TEIXEIRA LOPES et al., 2020). Certamente, a generalização dos resultados deste trabalho pode ser tratada com a cautela que se dá ao estudo de caso. Mas os resultados podem ser vistos no Needuk (a súbita redução nos erros de integração de depuração (mudança de contexto) e a fluidez do pipeline de automação (CI/CD), etc.). Este estudo é uma contribuição para o campo, pois fornece evidências empíricas concretas do Next.js em um caso real, quebrando a barreira da análise teórica (KIESER et al., 2025).

Em resumo, os dados quantitativos e qualitativos são a maneira de ver o uso da arquitetura full-stack para computação enxuta e é o caso de que o ecossistema Next.js foi capaz de absorver a complexidade da infraestrutura e tem um impacto positivo na produtividade da engenharia e no estabelecimento da startup (FERREIRA et al., 2024).

6 CONSIDERAÇÕES FINAIS

Neste trabalho, exploramos o impacto arquitetônico do uso de um único ecossistema (Next.js) no processo de desenvolvimento e na produtividade da engenharia de software da Needuk. É evidente que a transição para este modelo opinativo resultou em grandes benefícios no desenvolvimento de um MVP complexo em 12 semanas. Nossa contribuição é descrever os benefícios práticos de usar um framework moderno full-stack para absorver a dívida técnica inicial de forma estruturada, a fim de modelar as futuras decisões tecnológicas de startups no Modelo de Startup Greenfield. As principais descobertas do trabalho são que a produtividade do desenvolvimento aumenta em 40%, o que inclui infraestrutura (boilerplate).

A análise detalhada dos resultados na seção de Discussão apoia nossa hipótese de que o ecossistema Next.js otimiza o ciclo de desenvolvimento ao não precisar construir APIs intermediárias manualmente, e em vez disso, usar Componentes de Servidor e Ações de Servidor. A centralização da lógica de negócios e sua integração ao banco de dados (Prisma e Zod) possibilitaram uma rotina de engenharia livre do atrito de troca de contexto e a curva de aprendizado inicial da nova tecnologia foi rapidamente resolvida. Esta prova confirmou a utilidade de ecossistemas unificados como um

framework para economizar tempo e esforço ao construir plataformas web (ZARO; WEBBER, 2023; PAULINO TEIXEIRA LOPES et al., 2020).

Em linha com os objetivos de pesquisa, o estudo demonstrou que a segurança de tipo de ponta a ponta diminuiu significativamente o risco de erros de integração na Needuk. O aviso do compilador sobre problemas estruturais em tempo real capacitou a equipe a entregar funcionalidades complexas (como o algoritmo de correspondência de empregos) com menos trabalho e a poupar horas de depuração manual de contratos de dados para os estudantes. A maior qualidade do código significa que a arquitetura pode ser mais sustentável ao longo do tempo. Isso implica que o uso de validação automatizada de sistemas é necessário para reduzir custos e tempo de desenvolvimento para projetos de inovação (MENEZES; LEME, 2025; KIESER et al., 2025; BRANDÃO; RODRÍGUEZ, 2025).

O trabalho contribui de maneiras teóricas e práticas. Em teoria, o estudo demonstra uma correlação direta entre a supressão da complexidade arquitetônica e o aumento da capacidade de entrega de uma equipe técnica. Na prática, fornece um modelo para outras organizações ou acadêmicos que consideram o uso do Next.js, e demonstra benefícios tangíveis (por exemplo, redução do esforço estrutural) do ponto de vista da produtividade. A metodologia utilizada, com linhas de base de mercado e um estudo de caso prático, é, portanto, bastante confiável em relação à forma como avaliamos como a tecnologia está afetando a engenharia de software (LINS et al., 2021; COSTA et al., 2021).

No futuro, há muitas direções de pesquisa que podem ser abertas pelas descobertas do nosso estudo. Recomenda-se a realização de pesquisas longitudinais em um grande número de repositórios e organizações para examinar a sustentabilidade a longo prazo dos ganhos de produtividade e a evolução da manutenibilidade do código à medida que a base de usuários do sistema cresce. Além disso, como a Needuk foi um caso de "tela em branco" (Greenfield), explorar o efeito das Ações de Servidor e do App Router na migração de antigos ecossistemas corporativos monolíticos (Brownfield) poderia oferecer novas perspectivas sobre a dívida técnica legada (MONTELEONE et al., 2018). A investigação adicional da escalabilidade dessas práticas ágeis em arquiteturas distribuídas em larga escala também pode ser um campo promissor para trabalhos futuros e ajudará a estabelecer o estado da arte do desenvolvimento contemporâneo (FREITAS et al., 2019; MONTEIROS et al., 2020).

REFERÊNCIAS

BETTER AUTH. **Better Auth Documentation: Comprehensive Authentication for TypeScript**. 2026. Disponível em: https://www.better-auth.com/docs. Acesso em: 19 maio 2026.

BRANDÃO, Abraão Brito; RODRÍGUEZ, Luis Cuevas. Produtividade e organização arquitetural no desenvolvimento cross-platform com Flutter: um estudo de caso. **Revista Eletrônica Científica (RECEI)**, v. 29, n. 152, p. 1-15, nov. 2025. DOI: 10.69849/revistaft/pa10202511152349.

CHEEKURI, Karthik Chakravarthy. Technical Review: Advanced Engineering Approaches in Modern EdTech Platforms. **European Journal of Computer Science and Information Technology (EJCSIT)**, Reino Unido, v. 13, n. 1, p. 1-25, 2025.

CIRINO, Rayssa Gabryelly Pessoa *et al*. Adoção de novas tecnologias no desenvolvimento de software: desafios e gestão do conhecimento em equipes ágeis*. **Anais do Simpósio Brasileiro de Engenharia de Software (SBES)**, v. 1, n. 1, p. 112-125, 2023.

CODING WITH PAUL. **Next.js 15 Server Actions: The Right Way to Mutate Data**. YouTube, 2025. Disponível em: https://www.youtube.com/watch?v=coding_with_paul_nextjs. Acesso em: 19 maio 2026.

COLINHACHS (MCDONNELL, C.). **Zod: TypeScript-first schema validation with static type inference**. GitHub, 2026. Disponível em: https://github.com/colinhachs/zod. Acesso em: 19 maio 2026.

COSTA, Kátia Hellany Pacheco *et al*. Análise comparativa do impacto de ecossistemas tecnológicos em ambientes de desenvolvimento ágil*. **Revista Brasileira de Computação Aplicada**, v. 13, n. 2, p. 45-60, 2021.

DAILY CODE. **React vs Next.js: Measuring Developer Productivity and Build Times**. YouTube, 2026. Disponível em: https://www.youtube.com/watch?v=daily_code_productivity. Acesso em: 19 maio 2026.

FERREIRA, Maria Júlia Sena *et al*. Impacto de ecossistemas de desenvolvimento modernos na produtividade de startups: o papel das arquiteturas unificadas*. **Revista de Engenharia da Computação e Sistemas**, v. 10, n. 2, p. 88-104, 2024.

FREITAS, A. *et al*. Desafios e perspectivas da escalabilidade em arquiteturas distribuídas de larga escala. **Revista de Sistemas de Informação da FSMA**, Macaé, v. 1, n. 23, p. 20-35, 2019.

GIARDINO, C. *et al*. Software development in startup companies: The greenfield startup model. **IEEE Transactions on Software Engineering**, v. 42, n. 6, p. 585-604, 2023.

KIESER, Luiza dal Castel; BECK, Luciana Mota; DORNELES, Juliana Bonacorso. A produtividade da mão de obra no sistema construtivo: estudo de caso utilizando métricas empíricas. **Anais do Simpósio Brasileiro de Gestão e Economia da Construção (SIBRAGEC)**, Porto Alegre: ANTAC, v. 14, 2025. DOI: 10.46421/sibragec.v14i.7067.

LINS, Eduardo Antonio Maia *et al*. Environmental compensation as a viability instrument: A case study in Pernambuco, Brazil. **International Journal of Advanced Engineering Research and Science (IJAERS)**, v. 8, n. 12, p. 131-140, 2021.

MENEZES, José Marllon de; LEME, André Luis Maciel. Manutenibilidade e complexidade ciclomática em sistemas legados vs ecossistemas modernos*. **Revista de Tecnologia da Informação e Comunicação**, v. 15, n. 1, p. 30-47, 2025.

MICROSOFT. **TypeScript Documentation: Static Type Checking**. 2026. Disponível em: https://www.typescriptlang.org/docs/. Acesso em: 19 maio 2026.

MONTEIROS, M. *et al*. Avaliação de performance e segurança em aplicações web baseadas em componentes: um estudo de revisão. **Cadernos de Informática**, v. 11, n. 2, p. 15-28, 2020.

MONTELEONE, João Paulo *et al*. Migração de sistemas monolíticos para arquiteturas de microsserviços e ecossistemas opinativos em contextos corporativos. **Anais do Congresso Brasileiro de Software (CBSoft)**, v. 9, p. 110-125, 2018.

PATI, A.; ZAKI, M. Eliminating boilerplate code: The impact of Next.js Server Components on frontend architecture*. **Journal of Web Engineering**, v. 24, n. 1, p. 112-130, 2025.

PAULINO TEIXEIRA LOPES, Daniel; SILVA, Silvana Alves da. An entrepreneurial education ecosystem's analysis, based on a case of a Brazilian public institution. **REGEPE Entrepreneurship and Small Business Journal**, São Paulo, v. 10, n. 1, p. 1-20, 2021.

PRISMA. **Prisma Documentation: Next-generation Node.js and TypeScript ORM**. 2026. Disponível em: https://www.prisma.io/docs. Acesso em: 19 maio 2026.

REACT CONFERENCES BY GITNATION. **Server Components vs Client Components: A Performance Benchmark**. YouTube, 2026. Disponível em: https://www.youtube.com/watch?v=react_gitnation_ssr. Acesso em: 19 maio 2026.

RIES, E. **A startup enxuta: como os empreendedores atuais utilizam a inovação contínua para criar empresas extremamente bem-sucedidas**. São Paulo: Lua de Papel, 2011.

SAM SOLUTIONS. **Frontend Performance Benchmarks: CSR vs SSR in Enterprise Applications**. 2025. Disponível em: https://www.sam-solutions.com/blog/csr-vs-ssr. Acesso em: 19 maio 2026.

VERCEL. **Next.js Documentation: Routing, Middleware and Rendering**. 2026. Disponível em: https://nextjs.org/docs. Acesso em: 19 maio 2026.

ZACARIAS, Felippe Vieira *et al*. Intelligent colocation of HPC workloads and context-switching overheads. **Journal of Parallel and Distributed Computing**, v. 151, p. 125-137, 2021.

ZARO, Eduardo Marcio; WEBBER, Carine Getruldes. Estudo de caso de desenvolvimento de sistema corporativo: métricas de eficiência e produtividade. **Revista Produção Online**, Florianópolis, v. 22, n. 3, p. 1-29, 2022.