INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ENGENHARIA DE SOFTWARE NA PRÁTICA: RELATO DE EXPERIÊNCIA COMO DESENVOLVEDOR FULL-STACK

SAMUEL ANDRADE DE ARAÚJO

Cajazeiras
2025

SAMUEL ANDRADE DE ARAÚJO

ENGENHARIA DE SOFTWARE NA PRÁTICA: RELATO DE EXPERIÊNCIA COMO DESENVOLVEDOR FULL-STACK

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Me. Francisco Paulo de Freitas Neto.

Cajazeiras
2025

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

A663e Araújo, Samuel Andrade de.

Engenharia de software na prática : relato de experiência como desenvolvedor full-stack / Samuel Andrade de Araújo. – Cajazeiras, 2025.
34f. : il.

Trabalho de Conclusão de Curso (Tecnólogo em Análise de Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2025.

Orientador: Prof. Me. Francisco Paulo de Freitas Neto.

1. Engenharia de software. 2. Desenvolvimento de software. 3. Desenvolvimento Full-Stack.. 4. Experiência profissional. I. Instituto Federal de Educação, Ciência e Tecnologia da Paraíba. II. Título.

IFPB/CZ CDU: 004.41(043.2)

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

SAMUEL ANDRADE DE ARAÚJO

ENGENHARIA DE SOFTWARE NA PRÁTICA: RELATO DE EXPERIÊNCIA COMO DESENVOLVEDOR FULL-STACK

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Me. Francisco Paulo de Freitas Neto

Aprovada em: **09 de setembro de 2025.**

Prof. Me. Francisco Paulo de Freitas Neto - Orientador

Prof. Me. Fábio Abrantes Diniz - Avaliador

IFPB - Campus Cajazeiras

Prof. Dr. Fabio Gomes de Andrade - Avaliador

IFPB - Campus Cajazeiras

Documento assinado eletronicamente por:

- **Francisco Paulo de Freitas Neto**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 10/09/2025 16:10:47.
- **Fabio Abrantes Diniz**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 10/09/2025 16:45:23.
- **Fabio Gomes de Andrade**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 10/09/2025 16:56:00.

Este documento foi emitido pelo SUAP em 10/09/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código 764392
Verificador: b9847df58f
Código de Autenticação:

Dedico esse trabalho aos meus sobrinhos, Helena e Guilherme, por me lembrarem de minhas motivações.

AGRADECIMENTOS

Agradeço aos meus colegas de classe por sua ajuda nos momentos necessários, em especial Lúcio, José e João, pelo suporte fornecido. Também agradeço ao professor Diogo e ao professor Paulo pela orientação fornecida. Gostaria por fim de expressar minha sincera gratidão a Deus, Pai, Filho e Espírito Santo, pela força, sabedoria e bênçãos concedidas ao longo deste percurso.

"Qualquer tolo consegue escrever código que um computador entende. Bons programadores escrevem código que humanos conseguem entender."

Martin Fowler

RESUMO

A crescente transformação digital consolidou o *software* como um elemento central para a inovação em diversos setores, elevando o profissional de desenvolvimento a uma posição estratégica. Este trabalho tem como objetivo relatar a minha experiência profissional adquirida no processo de desenvolvimento de sistemas na empresa CEA, utilizando como metodologia o estudo de caso. A análise detalha as atividades, os desafios e as tecnologias empregadas na função de desenvolvedor *full-stack*, focando na concepção de soluções para setores especializados, como o da saúde, que demandam a gestão complexa e centralizada de dados. O processo de desenvolvimento é pautado por práticas ágeis, com a aplicação do *framework Scrum*, e utiliza um conjunto de tecnologias modernas, incluindo *Node.js*, *NestJS*, *React*, *Prisma ORM*, *MySQL* e *Docker*. Ao final, o relato demonstra a aplicação prática dos conceitos de engenharia de *software* na construção de uma plataforma robusta, evidenciando as decisões técnicas que garantem a integridade e a acessibilidade das informações em um ambiente corporativo.

Palavras-chave: Engenharia de *Software*. Desenvolvimento *Full-Stack*. Estudo de Caso. Metodologias Ágeis. *React*. *Node.js*.

ABSTRACT

The growing digital transformation has consolidated *software* as a central element for innovation across various sectors, elevating the development professional to a strategic position. This work aims to report my professional experience acquired in the system development process at CEA, using the case study methodology. The analysis details the activities, challenges, and technologies employed in the role of *full-stack* developer, focusing on designing solutions for specialized sectors, such as healthcare, which demand complex and centralized data management. The development process is guided by agile practices, with the application of the *Scrum framework*, and employs a set of modern technologies, including *Node.js*, *NestJS*, *React*, *Prisma ORM*, *MySQL*, and *Docker*. In the end, the report demonstrates the practical application of *software* engineering concepts in building a robust platform, highlighting the technical decisions that ensure the integrity and accessibility of information in a corporate environment.

Keywords:Software Engineering. Full-Stack Development. Case Study. Agile Methodologies. React. Node.js.

LISTA DE FIGURAS

Figura 1 – Logotipo da CEA - Cristofoleti Eletronic Arts. 17
Figura 2 – SCRUM – Resumo do funcionamento do processo. 22
Figura 3 – Trello — Exemplo de quadro de tarefas. 23

LISTA DE QUADROS

Quadro 1 – Informações da Empresa 17

LISTA DE ABREVIATURAS E SIGLAS

ADS	Análise e Desenvolvimento de Sistemas
CBSE	Component-Based Software Engineering (Engenharia de Software Baseada em Componentes).
CEA	Cristofoleti Eletronic Arts
CRUD	Criar, Ler, Atualizar, Deletar (as quatro operações básicas de banco de dados).
HTML	HyperText Markup Language
IFPB	Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
LAMP	Linux, Apache, MySQL, PHP/Python/Perl (uma popular pilha de tecnologias para desenvolvimento web)
ORM	Object-Relational Mapper (Mapeador Objeto-Relacional)
PO	Product Owner (Dono do Produto)
SGBD	Sistema de Gerenciamento de Banco de Dados
SDK	Software Development Kit
SP	São Paulo
SPAs	Single-Page Applications (Aplicações de Página Única)
SQL	Structured Query Language (Linguagem de Consulta Estrurada)
UI	User Interface (Interface de Usuário)
WIP	Work in Progress (Trabalho em Progresso)

SUMÁRIO

1 INTRODUÇÃO 15

1.1 Objetivos 16

1.1.1 Objetivo Geral 16

1.1.2 Objetivos Específicos 16

1.2 Apresentação da Empresa 16

2 MÃE CORUJA 18

2.0.1 Requisitos Funcionais 18

3 PROCESSO DE DESENVOLVIMENTO 20

3.1 Metodologia e Ferramenta 20

3.1.1 SCRUM 21

3.1.2 Trello 22

4 FUNDAMENTAÇÃO TEÓRICA 24

4.1 Tecnologias Utilizadas 24

4.1.1 Docker e Docker Compose 24

4.1.2 NodeJS com NestJS 25

4.1.3 Prisma ORM 26

4.1.4 MySQL 27

4.1.5 React 29

5 RELATO DE EXPERIÊNCIA 31

5.1 Desafios Encontrados 31

5.2 Oportunidades de Aprendizado 33

6 CONSIDERAÇÕES FINAIS 34

REFERÊNCIAS 35

1 INTRODUÇÃO

O avanço acelerado da transformação digital reconfigurou, nas décadas recentes, o ciclo de vida dos serviços, desde sua idealização até a entrega final ao consumidor. Tal fenômeno ofereceu um panorama dualista composto tanto por benefícios disruptivos quanto por complexos desafios operacionais e estratégicos (CASTELLS, 2017). Nesse contexto, os profissionais de tecnologia foram elevados a uma posição estratégica, tornando-se os agentes necessários para fundamentar essa expansão. A aplicação de suas competências se tornou um requisito em múltiplos segmentos de mercado.

Dentre as diversas especialidades do setor de tecnologia, destaca-se a do desenvolvedor de *software*. Este profissional é o responsável por analisar e traduzir problemas de naturezas distintas em soluções computacionais, cujo resultado é materializado na forma de um produto de *software* (U.S. News & World Report L.P., 2022).

O objeto de estudo delimitado para esta análise é o processo de desenvolvimento de *software* no ambiente da empresa CEA. Os projetos concebidos e implementados na organização servem como exemplos práticos e representativos das atividades de um desenvolvedor *full-stack* na construção de soluções corporativas, desde o levantamento de requisitos até a implementação final.

Os sistemas desenvolvidos pela empresa frequentemente atendem a demandas de setores especializados, como o da saúde. Um desafio comum nesses cenários é a gestão de informações em ambientes complexos, como clínicas de reabilitação multidisciplinar. Nesses contextos, a descentralização e a falta de padronização no registro de dados podem dificultar o acompanhamento integrado de pacientes por diferentes especialistas, como psicólogos, fonoaudiólogos e terapeutas ocupacionais. A fragmentação das informações clínicas não apenas compromete a continuidade do cuidado, mas também impede a geração de dados estruturados que poderiam ser utilizados para aprimorar os protocolos de tratamento e a gestão da própria clínica.

A superação desses obstáculos passa, invariavelmente, pela adoção de tecnologias que centralizem e organizem o fluxo informacional. A implementação de um software especializado surge como uma resposta estratégica para garantir que a jornada do paciente seja documentada de forma coesa e acessível, transformando dados brutos em conhecimento clínico aplicável. Nesse sentido, a literatura acadêmica reforça

a importância de tais ferramentas. Segundo (MARIN et al., 2003), “a Tecnologia da Informação (TI) vem se tornando um componente essencial para a modernização do setor de saúde, viabilizando a integração de dados clínicos e administrativos. A implementação de sistemas de informação permite não apenas a otimização de processos, mas também a qualificação da assistência, ao fornecer subsídios para decisões mais seguras e bem-informadas”.

Como solução para essa lacuna informacional, o sistema foi arquitetado para centralizar e organizar o registro de avaliações, intervenções e protocolos terapêuticos. Seu objetivo principal é fornecer uma plataforma unificada que garanta a integridade e a acessibilidade dos dados clínicos. Para tal, a aplicação foi equipada com módulos de gestão de usuários com perfis de acesso distintos, ferramentas para a criação de prontuários digitais e funcionalidades para o registro detalhado de sessões, permitindo um acompanhamento preciso do progresso de cada paciente.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Descrever as atividades e o processo de desenvolvimento, bem como as tecnologias utilizadas, no exercício da função de desenvolvedor *full-stack* na empresa CEA Artes Eletrônicas.

1.1.2 Objetivos Específicos

São os objetivos específicos deste trabalho:

- Apresentar os requisitos e as tecnologias-chave adotadas nos projetos da CEA.
- Descrever o processo de adaptação ao trabalho como desenvolvedor *full-stack* em regime *home office*, destacando os desafios e soluções encontrados.
- Analisar o impacto das tecnologias utilizadas no desenvolvimento do projeto, contextualizando-o no ambiente de uma *software house*.

1.2 APRESENTAÇÃO DA EMPRESA

A CEA é uma *software house* especializada no desenvolvimento de soluções tecnológicas personalizadas. Embora sua base de atuação seja no Brasil, a empresa possui um portfólio de projetos com alcance global, incluindo países como Estados Unidos, Espanha, Coreia do Sul, Austrália e Angola, entre outros. O Quadro 1 apresenta

os dados institucionais da organização, enquanto a figura 1 representa o logotipo utilizado pela empresa.

Quadro 1 – Informações da Empresa

CNPJ	10761464000174
Nome da Empresa	CEA Sistemas Administrativos LTDA ME
Nome Fantasia	CEA Artes Eletrônicas
Endereço	Rua Butantã, 95, Pinheiros, São Paulo
Página Oficial	<https://www.cea.eti.br/>

Figura 1 – Logotipo da CEA - Cristofoleti Eletronic Arts.

Fonte: (ARTS, 2025)

Sediada em São Paulo, SP, a empresa conta com uma equipe multidisciplinar, composta por profissionais de diversas especialidades. A organização adota um modelo de trabalho flexível, com colaboradores atuando em regimes presencial e *home office*, estrutura que suporta e otimiza a aplicação de práticas de desenvolvimento ágil. Para a descrição deste relato, a atuação profissional no cargo de desenvolvedor abrangeu projetos para plataformas *web* e *mobile*, com responsabilidades no desenvolvimento *full-stack*, englobando tanto o *front-end* quanto o *back-end* das aplicações.

2 MÃE CORUJA

Como desenvolvedor *full-stack* na CEA, o sistema na qual esse relato é centrado é o Mãe Coruja. A concepção desta plataforma nasceu de uma necessidade crítica identificada em clínicas de reabilitação multidisciplinar por parte do cliente, onde a fragmentação das informações sobre o paciente comprometia a eficácia e a continuidade do tratamento. O desafio era claro: criar uma solução que centralizasse a jornada terapêutica de cada paciente, oferecendo uma visão unificada e acessível para todos os especialistas envolvidos, como psicólogos, fonoaudiólogos e terapeutas ocupacionais.

Minha experiência no desenvolvimento do Mãe Coruja foi uma jornada de ponta a ponta. Atuando com tecnologias como *Node.js* e *NestJS* no back-end, estruturei uma API robusta, capaz de gerenciar com segurança os dados sensíveis da área da saúde. A utilização do *Prisma ORM* e do *MySQL* foi fundamental para modelar um banco de dados relacional que garantisse a integridade e a consistência das informações clínicas. A arquitetura foi pensada para ser escalável e segura, com módulos específicos para a gestão de usuários, implementando perfis de acesso distintos para assegurar que cada profissional visualizasse apenas as informações pertinentes à sua área de atuação.

No front-end, utilizando *React*, o foco foi criar uma interface intuitiva e responsiva, que simplificasse a rotina dos profissionais. Desenvolvi funcionalidades-chave, como a criação de prontuários digitais dinâmicos e ferramentas para o registro detalhado das sessões terapêuticas. Isso permitiu que o progresso de cada paciente fosse acompanhado de forma precisa e visual. Todo o ambiente de desenvolvimento e produção foi orquestrado com *Docker*, o que padronizou a configuração e facilitou a implantação contínua de novas funcionalidades.

O trabalho no sistema Mãe Coruja foi mais do que um desafio técnico; foi a experiência de traduzir uma necessidade real do setor de saúde em uma solução tecnológica funcional e significativa. Ver a plataforma transformar dados brutos e dispersos em conhecimento clínico aplicável, melhorando a qualidade do cuidado oferecido aos pacientes, consolidou meu entendimento sobre o papel estratégico do *software* como ferramenta de transformação.

2.0.1 Requisitos Funcionais

Quanto aos requisitos da aplicação, são descritos a seguir:

- **RF01:** O sistema deve permitir o cadastro, edição e desativação de contas de

usuário para os profissionais da clínica.

- **RF02:** O sistema deve permitir a atribuição de diferentes perfis de acesso aos usuários (ex: Administrador, Psicólogo, Fonoaudiólogo, Terapeuta Ocupacional).
- **RF03:** O sistema deve restringir o acesso a funcionalidades e dados de pacientes com base no perfil do usuário logado, garantindo a confidencialidade das informações.
- **RF04:** O sistema deve possuir uma funcionalidade de autenticação segura (login e logout) para todos os usuários.
- **RF05:** O sistema deve permitir o cadastro completo de novos pacientes, incluindo informações pessoais, de contato e dados demográficos relevantes.
- **RF06:** O sistema deve fornecer uma funcionalidade de busca para localizar rapidamente o cadastro de um paciente.
- **RF07:** O sistema deve permitir a visualização e a edição das informações cadastrais de um paciente existente por usuários autorizados.
- **RF08:** O sistema deve apresentar um prontuário digital unificado para cada paciente, exibindo todo o seu histórico de atendimentos e registros de forma centralizada.
- **RF09:** O sistema deve permitir que profissionais autorizados registrem novas sessões terapêuticas no prontuário do paciente.
- **RF10:** O formulário de registro de sessão deve permitir a inclusão de informações detalhadas, como: data, profissional responsável, tipo de intervenção, observações clínicas, progresso notado e anexos (se aplicável).
- **RF11:** O sistema deve permitir a visualização de todo o histórico de sessões de um paciente, de forma cronológica ou filtrada.
- **RF12:** O sistema deve ser capaz de gerar relatórios sobre a evolução do paciente, compilando dados dos registros das sessões em um período determinado.
- **RF13:** O sistema deve fornecer uma visualização clara (possivelmente gráfica ou em linha do tempo) do progresso do paciente para facilitar a análise clínica.

3 PROCESSO DE DESENVOLVIMENTO

3.1 METODOLOGIA E FERRAMENTA

Todo projeto de *software* nasce de uma necessidade específica: seja a correção de um defeito em um sistema existente, a adaptação de uma aplicação legada a novas condições de mercado, a expansão de suas funcionalidades ou a criação de um produto inteiramente novo (PRESSMAN; MAXIM, 2016). Nesse contexto, a Engenharia de Software emerge como a disciplina que estrutura os esforços para solucionar o problema-alvo por meio da concepção de um artefato de *software*, enfrentando desafios que transcendem a simples programação.

A construção de um *software* de qualidade, portanto, exige a aplicação de um processo de desenvolvimento bem definido. Tal processo é fundamental para assegurar que o produto final não apenas funcione corretamente, mas que também satisfaça plenamente os objetivos estratégicos e as necessidades dos usuários que motivaram sua criação. Segundo a teoria clássica da área (SOMMERVILLE, 2011), esse ciclo de vida do *software* envolve uma sequência de fases — como levantamento de requisitos, análise, projeto, codificação, testes e implantação — cuja execução ordenada visa mitigar os riscos e os altos custos associados a falhas em etapas tardias do desenvolvimento.

Contudo, os modelos de processo mais tradicionais, conhecidos como prescritivos ou planejados (a exemplo do modelo em cascata), mostraram-se rígidos e pouco adaptáveis a cenários de alta incerteza. Caracterizados por um planejamento extenso e inicial, esses modelos tratam os requisitos como um escopo fechado, tornando as mudanças de curso dispendiosas e complexas.

Como resposta a essa rigidez, surgiu o Movimento Ágil, formalizado em 2001 por meio do Manifesto Ágil. Sua filosofia rompe com a previsibilidade estrita dos modelos tradicionais ao valorizar "indivíduos e interações mais que processos e ferramentas"e "responder a mudanças mais que seguir um plano". Em vez de planejamentos monolíticos, o desenvolvimento ágil propõe ciclos curtos e iterativos (*sprints*), entregas de valor graduais e uma colaboração contínua com o cliente. Essa abordagem permite que as equipes se adaptem a novos contextos e diminuam drasticamente os custos e o tempo associados a mudanças de requisitos, mesmo que imprevistas.

Alinhada a essa filosofia, a empresa adota o *Scrum* como *framework* metodológico para a gestão e o desenvolvimento de seus projetos. O *Scrum* organiza o

trabalho em ciclos iterativos, promovendo transparência e adaptação contínua. Para a gestão visual das tarefas e o acompanhamento do fluxo de trabalho dentro de cada ciclo, a equipe utiliza a ferramenta *Trello*, que opera com base nos princípios do sistema *Kanban*.

3.1.1 SCRUM

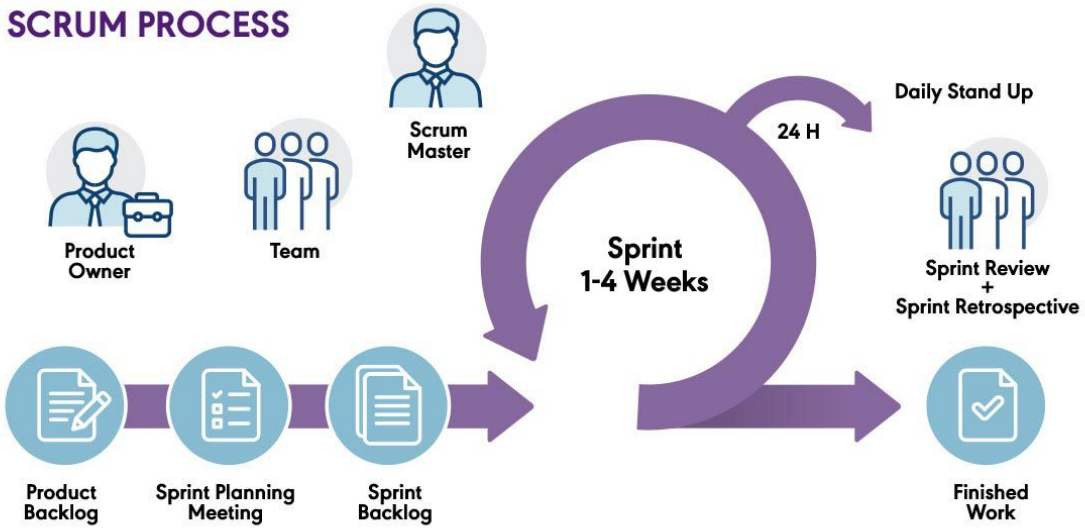
O *Scrum* não é uma metodologia prescritiva, conforme podemos ver em (SCHWABER; SUTHERLAND, 2020), mas sim um *framework* para o desenvolvimento e a manutenção de produtos complexos. Sua estrutura é fundamentada no empirismo, valorizando a transparência, a inspeção e a adaptação para otimizar a previsibilidade e controlar os riscos. O desenvolvimento é organizado em ciclos iterativos e incrementais chamados *Sprints*, que são eventos com duração fixa (*time-boxed*), geralmente de uma a quatro semanas, ao final dos quais uma versão funcional e potencialmente utilizável do produto é entregue.

O ponto de partida de todo o processo é o *Product Backlog*. Este não é apenas uma lista de tarefas, mas um artefato dinâmico e priorizado que contém todas as funcionalidades, requisitos, melhorias e correções desejadas para o produto. A gestão do *Product Backlog* é de responsabilidade do *Product Owner (PO)*, papel que representa os interesses dos *stakeholders* (as partes interessadas no projeto) e garante que a equipe de desenvolvimento esteja trabalhando nos itens que geram o maior valor para o negócio.

Cada *Sprint* se inicia com uma reunião de planejamento, a *Sprint Planning*, na qual a equipe de desenvolvimento seleciona um conjunto de itens do topo do *Product Backlog* para serem desenvolvidos durante aquele ciclo. Esses itens selecionados formam o *Sprint Backlog*, que é um plano detalhado de como o time transformará os requisitos em um incremento funcional do produto.

Durante a execução do *Sprint*, a equipe realiza reuniões diárias e curtas, as *Daily Scrums*, para sincronizar o trabalho e identificar impedimentos. Ao final do ciclo, ocorrem dois eventos cruciais: a *Sprint Review*, na qual o incremento do produto é apresentado aos *stakeholders* para obter *feedback*; e a *Sprint Retrospective*, uma reunião interna na qual a equipe reflete sobre o processo de trabalho e identifica pontos de melhoria para o próximo *Sprint*. Essa estrutura de eventos garante um ciclo contínuo de *feedback* e aprimoramento, alinhado aos princípios do Movimento Ágil. O framework SCRUM, portanto, organiza o trabalho em ciclos e cerimônias bem definidas. Para melhor visualização, a figura 2 a seguir demonstra a condução de todo esse processo, ilustrando como os papéis, artefatos e eventos se interligam

Figura 2 – SCRUM – Resumo do funcionamento do processo.



Fonte: (PM-PARTNERS, 2024)

É fundamental ressaltar, contudo, que a essência do desenvolvimento ágil reside na capacidade de adaptação. A aplicação estrita e dogmática de um *framework*, sem considerar o contexto do projeto, pode paradoxalmente comprometer a agilidade ao engessar o processo. Nesse sentido, a prática observada na CEA reflete uma maturidade na aplicação dos princípios ágeis: o *framework Scrum* é utilizado como uma base, mas suas práticas são constantemente adaptadas para se moldar às necessidades específicas de cada projeto e da equipe. Portanto, pode-se afirmar que a empresa aplica um modelo customizado do *Scrum*, mantendo-se fiel ao valor central do movimento ágil, que é a flexibilidade e a resposta a mudanças.

3.1.2 Trello

Para a gestão visual das tarefas e o acompanhamento do fluxo de trabalho, a equipe utiliza a ferramenta *Trello*. Trata-se de uma plataforma de colaboração visual projetada para organizar projetos e tarefas em um formato intuitivo e flexível. A principal força da ferramenta reside em sua simplicidade, que digitaliza o conceito do quadro *Kanban*, permitindo que as equipes visualizem o progresso do trabalho de forma clara e compartilhada (ATLASSIAN, 2025).

A estrutura do *Trello* é baseada em três componentes principais:

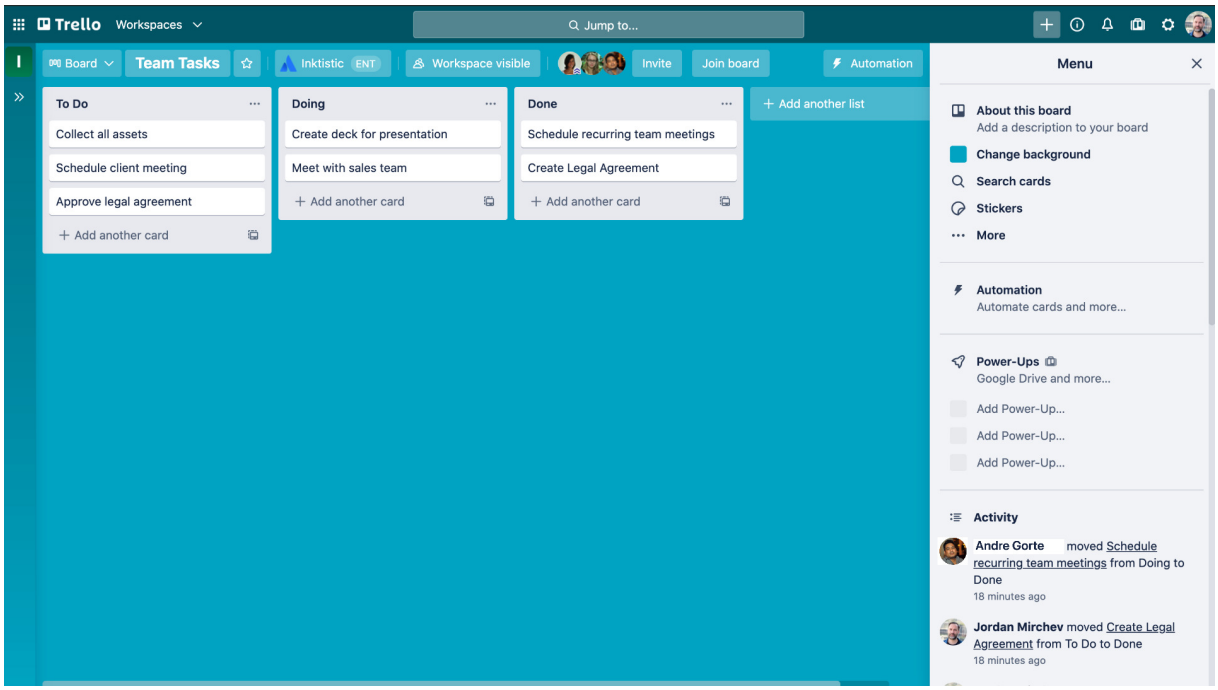
- **Quadros (*Boards*)**: Um quadro geralmente representa um projeto, um produto ou

um fluxo de trabalho específico. É o espaço principal onde todas as informações relacionadas a esse escopo são organizadas.

- **Listas (*Lists*)**: Dentro de um quadro, as listas representam as diferentes etapas ou estágios de um processo. A configuração mais comum segue o fluxo *Kanban* básico: "A Fazer"(*To Do*), "Em Andamento"(*Doing*) e "Concluído"(*Done*).
- **Cartões (*Cards*)**: Os cartões são as unidades fundamentais de trabalho, representando tarefas individuais. Eles são movidos entre as listas à medida que progridem no fluxo de trabalho. Cada cartão pode conter informações detalhadas, como descrições, *checklists*, prazos, anexos e comentários, servindo como um *hub* central para a comunicação sobre aquela tarefa específica.

Essa abordagem visual e baseada em arrastar e soltar (*drag-and-drop*) torna o *Trello* uma implementação digital direta do método *Kanban*, um sistema de gestão focado em otimizar o fluxo de trabalho e limitar o trabalho em progresso (*Work in Progress - WIP*). Ao visualizar todas as tarefas e seu *status* atual, as equipes podem identificar gargalos, gerenciar prioridades e manter um ritmo de entrega constante e sustentável, alinhado aos princípios de eficiência do desenvolvimento ágil (ANDERSON, 2011). Um exemplo claro de uso da ferramenta pode ser visto na figur 3, logo a seguir.

Figura 3 – Trello — Exemplo de quadro de tarefas.



Fonte: (TRELLO, 2025)

4 FUNDAMENTAÇÃO TEÓRICA

4.1 TECNOLOGIAS UTILIZADAS

Embora o universo de tecnologias para desenvolvimento de *software* seja amplo, a especialização em um conjunto coeso de ferramentas permite que as equipes atinjam maior produtividade e qualidade. Alinhado a essa prática, os projetos desenvolvidos na empresa contam com base em uma *stack* tecnológica bem definida, que aproveita a expertise da equipe para solucionar os desafios do projeto. As tecnologias que compõem essa arquitetura são apresentadas a seguir:

- *Docker* e *Docker Compose*
- *Node.js* com *NestJS*
- *Prisma ORM*
- *MySQL*
- *React*

4.1.1 Docker e Docker Compose

Um dos desafios centrais no desenvolvimento de *software* moderno é garantir que uma aplicação se comporte de maneira consistente em diferentes ambientes, desde a máquina do desenvolvedor até os servidores de produção. Para solucionar esse problema, consolidou-se o paradigma da conteinerização, uma abordagem de virtualização em nível de sistema operacional que empacota uma aplicação e todas as suas dependências — como bibliotecas, binários e arquivos de configuração — em uma unidade isolada e portátil chamada contêiner (TURNBULL, 2014).

Diferente das máquinas virtuais (VMs) tradicionais, que virtualizam um sistema operacional inteiro, os contêineres compartilham o *kernel* do sistema operacional do hospedeiro (*host*), tornando-os extremamente leves, rápidos para iniciar e eficientes no uso de recursos.

A tecnologia que popularizou e se tornou o padrão de fato para a conteinerização é o *Docker*. Trata-se de uma plataforma de *software* aberta que automatiza a implantação, o dimensionamento e a gestão de aplicações dentro de contêineres. O ecossistema *Docker* é baseado em alguns conceitos-chave:

- ***Dockerfile***: É um arquivo de texto que contém um conjunto de instruções para a construção de uma imagem *Docker*. Ele define o ambiente da aplicação, desde o sistema operacional base até a instalação de dependências e a cópia dos arquivos da aplicação.
- **Imagem (*Image*)**: É um pacote executável, leve e autônomo, que inclui tudo o que é necessário para executar uma aplicação. As imagens são criadas a partir de um *Dockerfile* e servem como um *template* para a criação de contêineres.
- **Contêiner (*Container*)**: É a instância em tempo de execução de uma imagem. É a unidade onde a aplicação é efetivamente executada, de forma isolada do ambiente hospedeiro e de outros contêineres.

Enquanto o *Docker* é excelente para gerenciar contêineres individuais, aplicações complexas frequentemente são compostas por múltiplos serviços que precisam interagir entre si (por exemplo, uma aplicação *web*, um banco de dados e um serviço de *cache*). Gerenciar o ciclo de vida de cada um desses contêineres manualmente seria complexo e propenso a erros. Para solucionar essa questão, foi criado o *Docker Compose*, uma ferramenta que permite definir e executar aplicações *Docker* multi-contêiner. Com um único arquivo de configuração no formato *YAML* (*docker-compose.yml*), o desenvolvedor pode descrever todos os serviços que compõem a aplicação, suas configurações de rede, volumes de dados e dependências. A partir desse arquivo, com um único comando (*docker-compose up*), é possível construir e iniciar todo o ambiente da aplicação de forma orquestrada e reprodutível (DOCKER INC., 2025).

4.1.2 NodeJS com NestJS

Historicamente, a linguagem *JavaScript* era executada exclusivamente no lado do cliente (*client-side*), dentro dos navegadores *web*. Essa limitação foi superada com o surgimento do *Node.js*, um ambiente de execução (*runtime environment*) que permitiu que o *JavaScript* fosse utilizado para construir o lado do servidor (*server-side*) de uma aplicação.

O *Node.js* não é uma linguagem de programação, mas sim uma plataforma construída sobre o motor *V8* do Google Chrome, o mesmo que executa o *JavaScript* nos navegadores. Sua principal característica é a arquitetura orientada a eventos e o modelo de entrada/saída (*I/O*) não bloqueante (*non-blocking*). Isso significa que o *Node.js* é capaz de lidar com um grande número de conexões simultâneas de forma eficiente, delegando operações demoradas (como consultas a banco de dados ou leituras de arquivos) e continuando a processar outras requisições. Essa arquitetura o torna ideal

para a construção de aplicações de rede escaláveis, como *APIs*, microserviços e sistemas em tempo real (TILKOV; VINOSKI, 2010).

Embora o *Node.js* seja extremamente poderoso, sua flexibilidade e natureza minimalista podem levar a uma falta de estrutura em projetos de grande escala, resultando em desafios de manutenção e escalabilidade. Para resolver essa questão, surgiram *frameworks* opinativos que fornecem uma arquitetura robusta sobre o *Node.js*.

O *NestJS* é um desses *frameworks*, destacando-se por ser progressivo e utilizar *TypeScript* por padrão, adicionando tipagem estática e os recursos mais recentes do *ECMAScript* ao *JavaScript*. Fortemente inspirado na arquitetura do *Angular*, o *NestJS* organiza o código em uma estrutura modular e bem definida, baseada em três componentes principais:

- **Módulos (*Modules*)**: Encapsulam um conjunto de funcionalidades relacionadas, organizando a aplicação em blocos coesos.
- **Controladores (*Controllers*)**: São responsáveis por receber as requisições HTTP e delegar a lógica de negócio para os serviços.
- **Serviços (*Services*)**: Contêm a lógica de negócio da aplicação, abstraindo as regras e a comunicação com fontes de dados, como bancos de dados.

Ao fornecer uma arquitetura sólida e pronta para uso, que promove princípios de *design* como Injeção de Dependência e Inversão de Controle, o *NestJS* permite que os desenvolvedores construam aplicações *back-end* altamente testáveis, escaláveis e de fácil manutenção, combinando a eficiência do *Node.js* com a organização de *frameworks* de nível empresarial (NESTJS, 2025).

4.1.3 Prisma ORM

A interação entre a lógica da aplicação e o banco de dados é uma das operações mais críticas e complexas no desenvolvimento de *software*. Tradicionalmente, essa comunicação exige a escrita manual de consultas *SQL*, o que pode ser repetitivo, propenso a erros e difícil de manter, especialmente em linguagens de programação orientadas a objetos. Para abstrair essa complexidade, surgiram os *ORMs (Object-Relational Mappers)*, ferramentas que mapeiam os modelos de dados da aplicação para as tabelas do banco de dados (WIDYARTO, 2023).

Nesse ecossistema, o *Prisma* se destaca como um *ORM* de nova geração para *Node.js* e *TypeScript*. Ele vai além dos *ORMs* tradicionais ao fornecer um conjunto de

ferramentas que melhora drasticamente a segurança e a produtividade do desenvolvedor ao interagir com o banco de dados (PRISMA, 2025). Sua arquitetura é composta por três componentes principais:

- ***Prisma Schema***: É o coração do *Prisma*. Trata-se de um arquivo declarativo (*schema.prisma*) que serve como a única fonte de verdade para o esquema do banco de dados e os modelos de dados da aplicação. Nele, o desenvolvedor define os modelos, seus campos e os relacionamentos entre eles de forma intuitiva, independentemente do tipo de banco de dados utilizado (*SQL* ou *NoSQL*).
- ***Prisma Client***: É um construtor de consultas (*query builder*) auto-gerado, totalmente seguro em tipos (*type-safe*), customizado para o *schema* definido. Após a definição dos modelos, o *Prisma Client* é gerado com um comando e fornece uma *API* programática para todas as operações de banco de dados (*CRUD* - Criar, Ler, Atualizar, Deletar). Sua principal vantagem é a integração nativa com o *TypeScript*, oferecendo autocompletar para as consultas e garantindo que erros de digitação ou de lógica sejam capturados em tempo de compilação, e não em tempo de execução.
- ***Prisma Migrate***: É a ferramenta de migração de banco de dados do *Prisma*. Com base nas alterações feitas no arquivo *schema.prisma*, o *Prisma Migrate* gera e aplica automaticamente os arquivos de migração *SQL* necessários para atualizar o esquema do banco de dados de forma segura e versionada. Isso simplifica a evolução da estrutura do banco de dados ao longo do ciclo de vida do projeto.

Ao combinar esses componentes, o *Prisma* oferece um fluxo de trabalho coeso e seguro para o desenvolvimento de aplicações, permitindo que os desenvolvedores escrevam consultas de forma mais rápida, com menos erros e com a confiança de que o código da aplicação está sempre sincronizado com a estrutura do banco de dados.

4.1.4 MySQL

Toda aplicação dinâmica necessita de um sistema para armazenar, gerenciar e recuperar dados de forma persistente e organizada. Para isso, utilizam-se os Sistemas de Gerenciamento de Banco de Dados (SGBDs). Dentre os modelos de SGBDs, o mais consolidado e amplamente utilizado é o modelo relacional.

O modelo de banco de dados relacional, proposto por Edgar F. Codd, fundamenta-se na teoria dos conjuntos e na lógica de predicados para organizar os dados em tabelas, também chamadas de relações. Cada tabela é composta por linhas (tuplas),

que representam um registro, e colunas (atributos), que representam as características desse registro. A principal força desse modelo é a sua capacidade de garantir a consistência e a integridade dos dados por meio de mecanismos como chaves primárias, chaves estrangeiras e restrições (*constraints*), minimizando a redundância e assegurando a precisão das informações (CODD, 1970).

O *MySQL* é um dos Sistemas de Gerenciamento de Banco de Dados relacionais de código aberto mais populares do mundo. Originalmente desenvolvido pela MySQL AB e atualmente mantido pela *Oracle*, ele se tornou um pilar fundamental de inúmeras aplicações *web*, sendo um componente central da popular pilha de tecnologia *LAMP (Linux, Apache, MySQL, PHP/Python/Perl)*.

As principais características do *MySQL* incluem: Confiabilidade e Desempenho: O *MySQL* é reconhecido por sua robustez, estabilidade e alta performance, sendo capaz de lidar com grandes volumes de dados e altas cargas de transações. Flexibilidade: Embora seja um SGBD relacional, o *MySQL* evoluiu para suportar diferentes motores de armazenamento (*storage engines*), como o *InnoDB* (que oferece transações *ACID* e chaves estrangeiras) e o *MyISAM*, permitindo que os desenvolvedores otimizem o banco de dados para diferentes casos de uso. Amplo Ecossistema: Por sua longa trajetória e popularidade, o *MySQL* possui uma vasta comunidade, documentação extensa e compatibilidade com praticamente todas as linguagens de programação e *frameworks* do mercado.

- **Linguagem *SQL***: A interação com o banco de dados é realizada por meio da *SQL (Structured Query Language)*, a linguagem padrão para consulta e manipulação de dados em SGBDs relacionais.
- **Confiabilidade e Desempenho**: O *MySQL* é reconhecido por sua robustez, estabilidade e alta performance, sendo capaz de lidar com grandes volumes de dados e altas cargas de transações.
- **Flexibilidade**: Embora seja um SGBD relacional, o *MySQL* evoluiu para suportar diferentes motores de armazenamento (*storage engines*), como o *InnoDB* (que oferece transações *ACID* e chaves estrangeiras) e o *MyISAM*, permitindo que os desenvolvedores otimizem o banco de dados para diferentes casos de uso.
- **Amplo Ecossistema**: Por sua longa trajetória e popularidade, o *MySQL* possui uma vasta comunidade, documentação extensa e compatibilidade com praticamente todas as linguagens de programação e *frameworks* do mercado.

Devido a essas características, o *MySQL* se estabelece como uma escolha sólida e confiável para a camada de persistência de dados, garantindo a integridade e a disponibilidade das informações gerenciadas pela aplicação.

4.1.5 React

O desenvolvimento de interfaces de usuário (*User Interfaces - UI*) para a *web* evoluiu de manipulações imperativas diretas do *DOM* para paradigmas mais abstratos e gerenciáveis. Nesse contexto, o *React* emergiu como uma tecnologia seminal, propondo uma abordagem declarativa para a construção de *UIs*, fundamentada em princípios da programação funcional e da engenharia de *software* baseada em componentes.

Desenvolvido pela *Meta* (anteriormente *Facebook*), o *React* é uma biblioteca *JavaScript* cujo objetivo principal é a criação de interfaces de usuário a partir de um modelo de componentes. Sua inovação não reside em novas funcionalidades para a *web*, mas na aplicação de paradigmas computacionais que simplificam o gerenciamento de estado e a renderização da *UI* (BANKER; VADLAMANI, 2020).

Os pilares teóricos que sustentam a arquitetura do *React* são:

- **O Paradigma Declarativo**: *UI* como uma Função de Estado A ideia central do *React* é que a interface do usuário é uma projeção do estado da aplicação. Em vez de escrever código imperativo para manipular o *DOM* (ex: "adicione este elemento", "remova aquela classe"), o desenvolvedor descreve declarativamente como a *UI* deve ser para um determinado estado. Quando o estado muda, o *React* se encarrega de atualizar o *DOM* para refletir a nova realidade. Esse modelo pode ser representado pela fórmula *UI = f(state)*, um princípio derivado da programação funcional, que torna o comportamento da aplicação mais previsível e fácil de depurar, pois a *UI* se torna um resultado determinístico de seus dados (ARMSTRONG, 2017).
- **Arquitetura baseada em Componentes**: O *React* adota integralmente os princípios da Engenharia de Software Baseada em Componentes (*CBSE - Component-Based Software Engineering*). A *UI* é decomposta em componentes — unidades independentes, encapsuladas e reutilizáveis que gerenciam seu próprio estado e lógica. Essa modularidade permite a composição de interfaces complexas a partir de blocos de construção simples, promovendo a coesão e o baixo acoplamento, que são características essenciais para a manutenibilidade e escalabilidade de sistemas de *software* (SZYPERSKI, 2002).

- ***Virtual DOM* e o Algoritmo de Reconciliação**: A abordagem declarativa seria computacionalmente inviável se, a cada mudança de estado, toda a *UI* fosse recriada no *DOM* real, uma vez que as operações de escrita no *DOM* são lentas. Para solucionar isso, o *React* implementa o *Virtual DOM*, uma representação da estrutura do *DOM* em memória. Quando o estado é alterado, uma nova árvore do *Virtual DOM* é criada. O *React* então executa um algoritmo de *diffing* (comparação), conhecido como Reconciliação, para identificar a diferença mínima entre a nova árvore e a anterior. Apenas essa diferença é aplicada ao *DOM* real, em um processo otimizado de atualizações em lote (*batching*), garantindo alta performance.

Graças à aplicação desses princípios, o *React* oferece um modelo de desenvolvimento robusto para a criação de *Single-Page Applications (SPAs)*, onde a complexidade do estado da *UI* é gerenciada de forma eficiente e escalável.

5 RELATO DE EXPERIÊNCIA

A transição do conhecimento teórico para a aplicação prática em um ambiente de desenvolvimento corporativo é marcada por uma série de desafios e oportunidades. Esta seção detalha os principais obstáculos encontrados e os aprendizados consolidados durante a atuação no ambiente de trabalho, que foram fundamentais para o desenvolvimento profissional.

Nesse sentido, a experiência corporativa, iniciada em 29 de abril na minha trajetória como desenvolvedor *full-stack*, funcionou como um **laboratório prático**, onde a teoria da engenharia de *software*, as metodologias de desenvolvimento e as arquiteturas de sistemas foram postas à prova diante de problemas e requisitos do mundo real. O que se apresenta a seguir, portanto, é uma **análise reflexiva** sobre essa jornada, buscando evidenciar como cada obstáculo técnico ou de processo se converteu em uma oportunidade valiosa para a consolidação de competências e para uma compreensão mais profunda da profissão.

5.1 DESAFIOS ENCONTRADOS

No campo técnico, a sólida base de conhecimento adquirida no ambiente acadêmico permitiu uma transição suave para a *stack* tecnológica utilizada. A adoção de *frameworks* opinativos como o *NestJS*, por exemplo, não constituiu uma barreira significativa, uma vez que conceitos como Injeção de Dependência e Módulos, já abordados na grade curricular, foram rapidamente assimilados. O foco, portanto, voltou-se para a aplicação prática da arquitetura em um contexto de produção. O principal desafio técnico emergiu com o crescimento da aplicação *React*, onde a reatividade e o gerenciamento de estado, embora simples em pequena escala, tornaram-se progressivamente mais complexos, demandando estratégias para otimizar renderizações e garantir a manutenibilidade do reuso de componentes. De forma semelhante, a adoção do *Docker* e *Docker Compose* representou uma oportunidade para aprofundar os fundamentos de virtualização, exigindo um estudo para além do básico para configurar interações complexas entre contêineres e garantir um ambiente de desenvolvimento padronizado.

Do ponto de vista metodológico e de processo, a aplicação das práticas ágeis em um contexto real também apresentou suas particularidades. A internalização do fluxo de trabalho do *Scrum*, com seus rituais de planejamento, reuniões diárias e retrospectivas, demandou uma adaptação à comunicação constante e à habilidade de decompor problemas complexos em tarefas estimáveis para um *sprint*. Essa dinâmica

foi intensificada pelo regime de trabalho *home office*, que impôs desafios de colaboração assíncrona, exigindo disciplina no uso de ferramentas como o *Trello* para garantir o alinhamento contínuo com a equipe e evitar retrabalho.

Enfrentei também um desafio considerável na arena de extração de dados. O que inicialmente parecia uma tarefa algorítmica simples de requisitar e analisar HTML, rapidamente se revelou um complexo jogo de estratégia. Deparei-me com barreiras como renderização de conteúdo via *JavaScript*, que exigiu o uso de ferramentas de automação de navegador como o *Puppeteer*, e mecanismos anti-scraping sofisticados, que me forçaram a desenvolver lógicas para rotação de *proxies* e simulação de comportamento humano. Essa experiência foi um aprendizado profundo sobre a arquitetura da web moderna e me ensinou que a extração de dados é uma tarefa de *reverse engineering* contínua, que demanda não apenas conhecimento técnico, mas também resiliência e uma capacidade de se adaptar constantemente às mudanças implementadas no sistema-alvo.

Minha jornada com o desenvolvimento móvel utilizando *React Native* foi marcada por um aprendizado profundo sobre a complexidade intrínseca do ecossistema *Android*. A promessa de um desenvolvimento multiplataforma ágil encontrou sua barreira na fragilidade da configuração do ambiente nativo. Problemas com o *Gradle*, incompatibilidades entre versões de *SDKs* e a depuração de erros que ocorriam apenas em dispositivos *Android* específicos consumiram um tempo significativo. Essa vivência me forçou a ir muito além do *JavaScript*, exigindo um mergulho no ambiente de desenvolvimento nativo do *Android Studio*, na análise de logs (*Logcat*) e na compreensão do processo de compilação. Aprendi, de forma prática, que um desenvolvedor *React Native* eficaz precisa ser também um detetive capaz de navegar e solucionar problemas nas camadas nativas de cada plataforma.

Uma das lições mais impactantes veio da minha participação em projetos de grande porte, onde a escala transforma desafios conhecidos em obstáculos de uma nova magnitude. A gestão do *débito técnico*, o alto acoplamento entre módulos e a simples dificuldade de manter uma visão coesa da arquitetura tornaram-se problemas diários. Compreendi na prática que, em projetos assim, a qualidade do código é indissociável da clareza da documentação, da robustez dos testes automatizados e, principalmente, da comunicação eficaz entre as equipes. A experiência me ensinou a valorizar *design patterns* que promovem o baixo acoplamento, a importância de um processo rigoroso de *code review* e a necessidade de planejar estrategicamente a evolução da arquitetura para garantir que o sistema continue sustentável e escalável a longo prazo.

Finalmente, a etapa mais crítica do aprendizado envolveu a capacidade de solucionar problemas de forma autônoma. A depuração de sistemas integrados, com múltiplas camadas (*front-end*, *back-end*, banco de dados), exigiu o desenvolvimento de uma abordagem sistemática para identificar a causa raiz dos problemas. Além disso, a habilidade de converter um requisito de negócio abstrato em uma solução técnica funcional e bem arquitetada consolidou-se como um aprendizado contínuo, envolvendo não apenas a codificação, mas a capacidade de questionar, clarificar e propor soluções alinhadas aos objetivos do produto.

5.2 OPORTUNIDADES DE APRENDIZADO

Cada desafio superado representou uma oportunidade de crescimento e consolidação de competências. A superação dos obstáculos técnicos, como a escalabilidade em *React*, proporcionou um entendimento aprofundado sobre arquitetura de *software* e a importância de decisões de *design* para a sustentabilidade de um projeto a longo prazo. Isso se traduziu na habilidade de aplicar padrões de projeto e estratégias de gerenciamento de estado que equilibram *performance* e manutenibilidade. A necessidade de depurar sistemas complexos, por sua vez, acelerou o desenvolvimento da autonomia e de uma visão sistêmica, permitindo compreender o fluxo de dados de ponta a ponta — desde a interação do usuário na interface até a persistência no banco de dados — e como cada componente do ecossistema tecnológico se interconecta.

Do ponto de vista metodológico, a vivência em um ambiente ágil e remoto evidenciou a importância de habilidades interpessoais (*soft skills*). A comunicação clara, a colaboração e a autogestão deixaram de ser conceitos abstratos e se mostraram ferramentas cruciais para o sucesso do projeto, tão importantes quanto a proficiência técnica. A experiência profissional, portanto, não apenas validou o conhecimento acadêmico, mas o enriqueceu com uma perspectiva prática, transformando desafios em aprendizados valiosos para a carreira de um desenvolvedor de *software*.

6 CONSIDERAÇÕES FINAIS

Este trabalho se propôs a relatar a experiência profissional na função de desenvolvedor de *software*, utilizando um estudo de caso para ilustrar a aplicação prática dos conceitos de engenharia de *software* em um ambiente corporativo. Ao longo dos capítulos, foi detalhado o processo de desenvolvimento, desde a concepção de uma solução para uma necessidade de negócio específica até a implementação com o uso de tecnologias modernas e metodologias ágeis.

Constatou-se que a base teórica adquirida na formação acadêmica se mostrou fundamental para a rápida assimilação de *frameworks* e tecnologias como *NestJS*, *React* e *Docker*. No entanto, a experiência prática revelou que os maiores desafios não residem apenas no domínio técnico, mas na aplicação desses conhecimentos para resolver problemas complexos em um sistema de larga escala. A escalabilidade da aplicação, a manutenibilidade do código e a depuração de um ecossistema integrado foram os pontos que mais exigiram aprofundamento e que, consequentemente, geraram os aprendizados mais significativos.

A vivência no processo de desenvolvimento ágil, pautado pelo *Scrum*, demonstrou que a proficiência técnica deve ser complementada por habilidades interpessoais robustas. A comunicação, a colaboração e a autogestão em um ambiente de trabalho remoto foram cruciais para o sucesso do projeto, evidenciando que o desenvolvimento de *software* é uma atividade intrinsecamente colaborativa.

Conclui-se, portanto, que a experiência profissional não apenas validou o conhecimento teórico, mas o enriqueceu, preenchendo a lacuna entre a teoria e a prática. Os desafios enfrentados foram essenciais para a consolidação de uma visão sistêmica e para o desenvolvimento da autonomia necessária para atuar como um desenvolvedor de *software* no cenário atual de transformação digital. O objetivo de descrever o processo de desenvolvimento foi alcançado, e os resultados reforçam a importância de uma formação sólida aliada à capacidade de adaptação e aprendizado contínuo.

REFERÊNCIAS

ANDERSON, D. J. **Kanban: Mudança Evolucionária de Sucesso para seu Negócio de Tecnologia**. São Paulo: Bookman, 2011.

ARMSTRONG, M. **Functional JavaScript: Introducing Functional Programming with Underscore.js**. Sebastopol, CA: O'Reilly Media, 2017.

ARTS, C. C. E. **Logotipo da CEA**. 2025. <https://www.cea.eti.br/>. Acesso em: 29 ago. 2025.

ATLASSIAN. **Trello**. 2025. [S.l.]. Disponível em: <https://trello.com>. Acesso em: 17 ago. 2025.

BANKER, H.; VADLAMANI, R. A comparative study of web application development using angular and react. In: **2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)**. Tirunelveli, India: IEEE, 2020. p. 822–827.

CASTELLS, M. **A Sociedade em Rede**. 18. ed. São Paulo: Paz e Terra, 2017.

CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, v. 13, n. 6, p. 377–387, 1970.

DOCKER INC. **Overview of Docker Compose**. S.l.: [s.n.], 2025. <https://docs.docker.com/compose/>. Acesso em: 17 ago. 2025.

MARIN, H. d. F.; TIRONI, M.; MORI, F. **Sistemas de Informação em Saúde**. São Paulo: Editora Atheneu, 2003.

NESTJS. **NestJS Documentation**. S.l.: [s.n.], 2025. <https://docs.nestjs.com>. Acesso em: 17 ago. 2025.

PM-PARTNERS. **The Agile Journey: A Scrum Overview**. 2024. <https://www.pm-partners.com.au/insights/the-agile-journey-a-scrum-overview/>. Acesso em: 29 ago. 2025.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016.

PRISMA. **Prisma Documentation**. S.l.: [s.n.], 2025. <https://www.prisma.io/docs>. Acesso em: 17 ago. 2025.

SCHWABER, K.; SUTHERLAND, J. **O Guia do Scrum: As Regras do Jogo**. 2020. Scrum.org. Disponível em: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: 17 ago. 2025.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

SZYPERSKI, C. **Component Software: Beyond Object-Oriented Programming**. 2. ed. Boston: Addison-Wesley, 2002.

TILKOV, S.; VINOSKI, S. **Node.js in Action**. Shelter Island, NY: Manning Publications, 2010.

TRELLO. **Trello 101: How to Use Trello Boards & Cards**. 2025. <https://trello.com/guide/trello-101#what-is-a-list>. Acesso em: 29 ago. 2025.

TURNBULL, J. **The Docker Book: Containerization is the new virtualization**. S.l.: James Turnbull, 2014.

U.S. News & World Report L.P. **Software Developer**. 2022. Acesso em: 17 ago. 2025. Disponível em: <https://money.usnews.com/careers/best-jobs/software-developer>.

WIDYARTO, B. **Building a Full-Stack Web Application with Next.js, Prisma, and PostgreSQL**. S.l.: Apress, 2023.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Cajazeiras - Código INEP: 25008978
Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

Entrega do TCC

Assunto:	Entrega do TCC
Assinado por:	Samuel Andrade
Tipo do Documento:	Relatório
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Samuel Andrade de Araujo**, **DISCENTE (202122010023) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 12/09/2025 16:17:29.

Este documento foi armazenado no SUAP em 12/09/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1607606
Código de Autenticação: 2a63436395