UNIVERSIDADE FEDERAL DO PAMPA – UNIPAMPA

ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO PARCIAL DA DISCIPLINA DE ESTÁGIO OBRIGATÓRIO

Identificação do aluno			
Nome:Michel Almeida da Silva			Matrícula: 2501570095
Endereço: LEODORO DIAS DA SILVA 445			
Cidade: ERECHIM		UF: RS	CEP:99704-634
E-mail: michelsilva.aluno@unipampa.edu.br			Telefone:54981191013
CPF:04051133079	RG: 8123355755		Órgão emissor: SSP RS

Modalidade de desenvolvimento das atividades
( ) Estágio Obrigatório ( ) Estágio Não obrigatório (X) Aproveitamento de Atividades Profissionais

Identificação da empresa		
Nome: RAIA DROGASIL S.A.		CNPJ: 61.585.865/0001-51
Endereço: Avenida Corifeu de azevedo Marques, 3097		
Cidade: SÃO PAULO	UF: SP	CEP:05339-900
E-mail:		Telefone: (11)3769-5678

Atividades desenvolvidas			
Período do relatório	Data de início:25/09/2025		Data de fim:13/10/2025
Carga horária de estágio	Diária:8h	Semanal:40h	Total:100h
As atividades desenvolvidas foram executadas dentro de um framework (estrutura de trabalho) de Metodologia Ágil, especificamente o Scrum. O trabalho foi organizado em Sprints (ciclos de desenvolvimento) com duração de uma semana cada Sprint. Minha atuação como Desenvolvedor Full-Stack (profissional que atua tanto no front-end quanto no back-end da aplicação) com perfil de liderança envolveu a participação ativa em todas as cerimônias ágeis (reuniões previstas pelo Scrum), a colaboração intensa com a equipe e a aplicação de princípios da Engenharia de Software para garantir entregas de valor contínuas e de alta qualidade. **Referencial Teórico**			

Deverias explicar o porquê destes conceitos estarem sendo explicados aqui.

As bases técnicas que nortearam a execução das atividades foram a adoção de quatro pilares fundamentais: uma arquitetura de microsserviços, a aplicação dos princípios SOLID, a utilização de um Design System e diferentes linguagens e frameworks. Essa combinação permitiu a entrega de produtos e funcionalidades de forma incremental, consistente e evolutiva, conforme preconizado pelo Manifesto Ágil. [3]

Scrum: É um dos frameworks mais populares para a implementação de desenvolvimento ágil. Ele é projetado para que equipes possam entregar valor ao cliente de forma incremental e iterativa. [1]

Engenharia de Software: É a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software. [2]

Arquitetura de Microsserviços: É um padrão de arquitetura onde uma aplicação complexa é composta por um conjunto de serviços pequenos e independentes, cada um responsável por uma capacidade de negócio específica e comunicando-se através de APIs. [4]

Princípios SOLID: É um acrônimo para cinco princípios de design de software que visam tornar o código mais compreensível, flexível e manutenível. [5]

- S - Single Responsibility Principle
- O - Open/Closed Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

Design System (Sistema de Design): É a fonte única da verdade que agrupa todos os elementos que permitem às equipes projetar e desenvolver um produto de forma coesa e escalável. [6]

O Pulso, Design System da RD Saúde: O "Pulso" materializa este conceito ao prover Tokens de Design (cores, tipografia), uma biblioteca de Componentes reutilizáveis e Documentação Técnica. Seu objetivo é "proporcionar uma base sólida para que designers e desenvolvedores possam criar produtos e experiências com qualidade, coesão e alinhamento", garantindo consistência e produtividade.

Node.js (Ambiente de Execução Back-end): É um ambiente de execução (runtime) de JavaScript assíncrono, construído sobre o motor V8 do Google Chrome. Ele permite a execução de código JavaScript no lado do servidor.

- Estrutura e Funcionamento: Seu principal diferencial é o modelo de I/O (Entrada/Saída) não bloqueante, orientado a eventos. Através do Event Loop, o Node.js consegue lidar com um grande número de conexões simultâneas de forma eficiente, delegando operações demoradas (como consultas a banco de dados ou chamadas de API) e executando outras tarefas enquanto aguarda a resposta. Isso o torna ideal para a construção de microsserviços e APIs de alta performance.

Next.js (Framework Web Front-end): É um framework React para a construção de aplicações web de produção. Ele estende o React com uma estrutura robusta para renderização, roteamento e otimizações de performance.

- Estrutura e Funcionamento: Next.js é conhecido por seu sistema de roteamento baseado em arquivos e suas múltiplas estratégias de renderização: SSR (Server-Side Rendering), que gera o HTML da página no servidor a cada requisição, ideal para SEO e

conteúdo dinâmico; e SSG (Static Site Generation), que gera o HTML em tempo de compilação (build), oferecendo performance máxima para páginas estáticas.

Metodologia Ágil: É uma abordagem iterativa para o gerenciamento de projetos e desenvolvimento de software que ajuda as equipes a entregar valor aos seus clientes de forma mais rápida e com menos dores de cabeça.

Código Limpo (Clean Code): É uma filosofia de desenvolvimento de software popularizada por Robert C. Martin ("Uncle Bob"). A premissa é que o código deve ser escrito de forma simples, direta e legível, como uma prosa bem escrita. Suas características incluem o uso de nomes significativos para variáveis e funções, funções pequenas com responsabilidades únicas e a minimização de complexidade.

Arquitetura Limpa (Clean Architecture): Também concebida por Robert C. Martin, é um modelo de design de software que impõe uma rigorosa separação de responsabilidades. A arquitetura é representada por círculos concêntricos, onde a regra principal é a Regra da Dependência: as dependências do código-fonte só podem apontar para dentro. Nada em um círculo interno pode saber qualquer coisa sobre um círculo externo. Isso cria um sistema:

Design Patterns (Padrões de Projeto): São soluções gerais e reutilizáveis para problemas que ocorrem com frequência no desenvolvimento de software. Eles não são códigos prontos, mas sim modelos ou descrições de como resolver um problema que pode ser usado em muitas situações diferentes. Popularizados pelo livro "Design Patterns: Elements of Reusable Object-Oriented Software" (do grupo "Gang of Four" ou GoF), eles são categorizados em:
Criação (Creational): Abstraem o processo de instanciação de objetos (ex: Factory, Singleton).
Estruturais (Structural): Descrevem como classes e objetos podem ser combinados para formar estruturas maiores e mais complexas (ex: Adapter, Decorator).
Comportamentais (Behavioral): Tratam da comunicação e da atribuição de responsabilidades entre objetos (ex: Strategy, Observer).

Arquitetura Monolítica: É o modelo tradicional de arquitetura de software, no qual uma aplicação é construída como uma única unidade coesa e indivisível. Todos os seus componentes (interface do usuário, lógica de negócio, acesso a dados) são fortemente acoplados e executados como um único serviço.

Chrome DevTools: É um conjunto de ferramentas de desenvolvimento web embutido no navegador Google Chrome. Permite inspecionar o DOM e CSS (painel Elements), depurar JavaScript (painel Sources), monitorar requisições de rede (painel Network) e analisar a performance de renderização da página (painel Performance).

Lighthouse: É uma ferramenta de auditoria automatizada de código aberto do Google, integrada ao DevTools. Ela avalia a qualidade de uma página web com base em cinco categorias principais: Performance, Acessibilidade, Boas Práticas, SEO e Progressive Web App (PWA).

Web Vitals: É uma iniciativa do Google para fornecer um conjunto unificado de métricas de qualidade, focadas em quantificar a experiência do usuário na web.

Core Web Vitals (CWV): São um subconjunto dos Web Vitals que o Google considera essenciais para todas as páginas. Eles medem três aspectos da experiência do usuário: carregamento, interatividade e estabilidade visual. Atualmente, os três pilares dos CWV são:

LCP (Largest Contentful Paint): Mede a performance de carregamento. É o tempo que o maior elemento de conteúdo (imagem ou bloco de texto) leva para se tornar visível na tela. Uma boa experiência deve ter um LCP de até 2,5 segundos.

CLS (Cumulative Layout Shift): Mede a estabilidade visual. Quantifica o total de saltos inesperados de layout que ocorrem durante o carregamento da página. Uma boa experiência deve ter um CLS de 0.1 ou menos.

INP (Interaction to Next Paint): Mede a interatividade. Avalia a latência de todas as interações do usuário com a página, reportando o pior valor. Uma boa experiência deve ter um INP abaixo de 200 milissegundos. Essas métricas são um fator relevante para o ranking de busca do Google.

React: É uma biblioteca JavaScript de código aberto, mantida pelo Facebook, para a construção de interfaces de usuário (UI). Seu principal paradigma é a criação de UIs de forma declarativa e baseada em componentes. Em vez de dizer ao navegador como manipular o DOM, o desenvolvedor declara como a UI deve se parecer em diferentes estados, e o React se encarrega de atualizar o DOM de forma eficiente através de um mecanismo chamado Virtual DOM.

Conceitos Fundamentais do React:

- Componentes: São peças de código isoladas e reutilizáveis que rendem uma parte da UI.

- Reatividade: A UI "reage" automaticamente a mudanças nos dados da aplicação (o "estado"). Quando o estado de um componente muda, o React automaticamente re-renderiza aquele componente e seus filhos para refletir a nova informação.

- Gerenciamento de Estados (State Management): O "estado" (state) é um objeto que armazena os dados de um componente que podem mudar ao longo do tempo. O React fornece mecanismos para gerenciar o estado local de um componente (Hook useState) e também para compartilhar estado entre múltiplos componentes (Hook useContext ou bibliotecas externas como Redux e Zustand).

- Hooks: São funções especiais que permitem serem chamadas nos recursos de estado e ciclo de vida do React a partir de componentes de função. useState para adicionar estado, useEffect para lidar com efeitos colaterais (como chamadas de API) e useContext para acessar dados globais são alguns dos Hooks mais utilizados.

- Roteamento (Routing): Em uma Single-Page Application (SPA) construída com React, o roteamento é o processo de navegar entre diferentes "páginas" sem recarregar o navegador. Isso é gerenciado por bibliotecas como o React Router ou por frameworks como o Next.js, que possuem um sistema de roteamento integrado.

Ecossistema Android (Nativo): É um sistema operacional móvel de código aberto baseado no Kernel Linux, mantido primariamente pelo Google. Sua natureza aberta permite que seja modificado e distribuído por diversos fabricantes de hardware, resultando em um ecossistema de dispositivos vastos e heterogêneos.

- Linguagens de Programação:Atualmente, Kotlin é a linguagem oficial e recomendada pelo Google. Kotlin é totalmente interoperável com Java, mas oferece vantagens como sintaxe mais concisa, segurança contra nulos (null safety) e funcionalidades modernas que aumentam a produtividade e a segurança do código. Anteriormente o desenvolvimento era prioritariamente realizado em Java.

- Arquitetura e Componentes: Uma aplicação Android é estruturada em componentes fundamentais, como Activities (representam uma tela da UI), Services (para operações em segundo plano), Broadcast Receivers (para responder a eventos do sistema) e Content Providers (para compartilhar dados).

- Gerenciamento de Build: O sistema de build oficial é o Gradle, uma ferramenta de automação que compila o código, gerência dependências de bibliotecas (via repositórios como Maven Central) e empacota a aplicação nos formatos APK ou AAB (Android App Bundle) para distribuição.

- Distribuição: A principal plataforma de distribuição é a Google Play Store.

Ecossistema iOS (Nativo): É o sistema operacional móvel proprietário da Apple, projetado para rodar exclusivamente em seus dispositivos (iPhone). É conhecido por seu ecossistema fechado e controlado, o que resulta em um alto padrão de consistência, segurança e performance.

- Linguagens de Programação: O desenvolvimento legado era feito em Objective-C. A linguagem moderna e recomendada é o Swift, projetada pela Apple com foco em segurança, velocidade e sintaxe expressiva. Swift e Objective-C são interoperáveis.

- Arquitetura e Componentes: O desenvolvimento iOS é baseado em um conjunto de frameworks poderosos:

- Gerenciamento de Dependências: O gerenciador de pacotes oficial e integrado ao Xcode é o Swift Package Manager (SPM). Ferramentas de terceiros, como o CocoaPods, ainda são amplamente utilizadas em projetos legados.

- Distribuição: A distribuição é feita exclusivamente pela Apple App Store, que possui um rigoroso processo de revisão para garantir a qualidade e a segurança dos aplicativos.

Ecossistema React Native (Multiplataforma):: É um framework para desenvolver aplicações móveis para Android e iOS usando uma única base de código em JavaScript e React. React Native envolve não apenas a criação de interfaces em JavaScript, mas também a capacidade de escrever módulos nativos (Java/Kotlin ou Swift/Objective-C) quando necessário para otimizar performance ou acessar APIs específicas do sistema operacional.

Atuação em tarefas:

---

1. Desenvolvimento de um MVP de um Sistema Web (Período: 30/09 a 25/11)
    - Detalhamento da Execução: O MVP está sendo construído e tem previsão de acabar ao longo do final das atividades previstas na disciplina.
    - Atuação em Tarefas
        - **Design e Implementação do API Gateway e do Contrato de Comunicação Inter-serviços.**
            - A simples criação de endpoints individuais em cada microsserviço levaria a um acoplamento forte com o front-end e a uma complexidade de gerenciamento. Aplicando os princípios de Design de Software, foi implementada uma API Gateway para desacoplamento entre regras de negócios da aplicação.
            - Execução Baseada em Engenharia de Software:
                1. Análise de Requisitos: Em colaboração com a equipe de front-end, mapeei os casos de uso das telas. Isso definiu os dados que o Gateway precisaria agregar.
                2. Design da Arquitetura: Desenhamos o Gateway como um serviço Node.js simples, cuja única responsabilidade (Princípio 'S' do SOLID) era receber requisições do cliente, orquestrar chamadas para os microsserviços internos e agregar as respostas. Isso desacoplou os clientes da topologia interna da nossa rede de serviços.
                3. Construção e Qualidade: Definimos um contrato de comunicação claro usando a especificação OpenAPI (Swagger). Isso permitiu que as equipes de front-end e back-end trabalhassem em paralelo, usando o contrato documentado como fonte da verdade. O Gateway também centralizou a lógica de autenticação de rotas e o tratamento de CORS, simplificando os serviços internos. A implementação seguiu padrões de código limpo para garantir a manutenibilidade futura.

2. Implementação de Componente de Vídeo (Período: 06/10 a 25/11)

- Detalhamento da Execução: O componente está em fase final de desenvolvimento, com a lógica central e os testes unitários concluídos. A etapa final, prevista para as próximas Sprints, é a integração do componente nas telas da aplicação.

- Atuação em Tarefas:
    - Design, Implementação e Otimização de um Player de Vídeo Reutilizável.
        - A criação de um player de vídeo envolve a construção de uma Máquina de Estados Finita (Finite State Machine - FSM) para gerenciar o ciclo de vida do player, a otimização de performance para evitar gargalos de renderização e a garantia de robustez através de uma suíte de testes completa.
        - Execução Baseada em Engenharia de Software:
          1. Design de Componentes e Lógica de Estado (Hook):

2. Seguindo o princípio de Separação de Responsabilidades e a filosofia da Arquitetura Limpa, a lógica de negócio foi completamente isolada da camada de apresentação (UI).
3. Custom Hook (useVideoPlayerState): Criei um custom hook que serve como o "cérebro" do componente. Ele encapsula toda a interação com a API da biblioteca de vídeo (ex: React Player) e gerencia a FSM interna do player.
    - Estados da FSM: O hook gerencia os seguintes estados: IDLE, LOADING, PLAYING, PAUSED, SEEKING, ENDED, ERROR.
    - API Exposta pelo Hook: O hook retorna um objeto bem definido para o componente de UI, contendo:
        - Estado (state): Um objeto com informações reativas como { isPlaying, progress, duration, isLoading, hasError }.
        - Ações (actions): Funções para controlar o player, como { handlePlay, handlePause, handleSeek, toggleMute }.

- Construção e Otimização de Performance:
    - A performance foi tratada como um requisito fundamental, especialmente em interações de alta frequência.
    - Evento onProgress: Este evento pode disparar múltiplas vezes por segundo, o que, se gerenciado ingenuamente com useState, causaria um excesso de re-renderização no React, sobrecarregando e resultando em uma UI "travada".
- Solução de Engenharia:
    1. Manipulação Direta do DOM para UI: A atualização visual da barra de progresso, que precisa ser fluida, foi desacoplada do estado do React. Utilizei o hook useRef para obter uma referência direta ao elemento da barra e atualizei seu estilo (transform: scaleX(...)) diretamente, bypassando o ciclo de renderização do React para essa animação específica.
    2. Throttling para o Estado React: O estado global do React (ex: o texto "01:32 / 05:00") ainda precisava ser atualizado, mas não com a mesma frequência. Implementei uma função de throttle (usando a biblioteca lodash, por exemplo) para garantir que o setState fosse chamado no máximo uma vez a cada 250ms, garantindo a atualização da informação sem sobrecarregar a aplicação.
- Memoização: Todos os sub-componentes da UI (ex: <PlayPauseButton />, <VolumeControl />) foram envolvidos em React.memo, e as funções de callback passadas como props foram estabilizadas com useCallback. Isso previne re-renderizações desnecessárias quando o estado pai muda, mas as props específicas daquele componente filho permanecem as mesmas.

- Testes Unitários do Hook useVideoPlayerState:
  A qualidade e a robustez do componente foram garantidas através de testes unitários focados na lógica de estado, implementados com Jest e React Testing Library. A abordagem foi testar o custom hook em total isolamento, simulando os eventos da biblioteca de vídeo e verificando as transições de estado e as ações executadas, na tabela 1, estão evidenciados os cenários de testes unitários abordados.

Tabela 1 - Testes unitários na Implementação de Componente de Vídeo

Objetivo	Verificação(Assert)
Garantir que o hook inicie no estado correto	O estado inicial isPlaying deve ser false, progress deve ser 0, isLoading deve ser true (ou false, dependendo da estratégia de autoplay).
Validar a mudança de estado ao simular eventos de play e pause.	Ao simular o evento onPlay da biblioteca, o estado isPlaying deve se tornar true. Ao simular onPause, deve se tornar falso.
Assegurar que o progresso e a duração do vídeo são refletidos no estado corretamente.	Ao simular um evento onProgress com { played: 0.5, loaded: 0.8 }, o estado progress deve ser atualizado para 0.5.
Verificar a transição para o estado ENDED.	Ao simular o evento onEnded, o estado isPlaying deve se tornar falso e um estado isEnded (se existir) deve ser true.
Garantir que o hook lide corretamente com erros do player.	Ao simular um evento onError, o estado hasError deve se tornar true, e isLoading deve ser falso.
Validar que a ação do usuário para buscar uma posição no vídeo é corretamente processada.	Ao chamar a ação handleSeek(0.75), o mock da função de seek da biblioteca de vídeo deve ser chamado com o argumento 0.75.
Verificar se a ação de mutar/desmutar o áudio funciona.	Chamar toggleMute deve alterar o estado isMuted de false para true, e vice-versa.

Fonte: Do próprio autor

Tarefa ainda não finalizada: Necessário a integração dentro das telas que farão o uso do componente. Esta próxima fase envolverá a passagem das props necessárias (como a URL do vídeo) e a conexão de callbacks (como onEnded) com a lógica de negócio da página que o contém.

3. Estudos de Viabilidade Técnica para Funcionalidades de Vídeos com interatividades.

- Detalhamento da Execução: Os estudos foram focados em gerar conhecimento técnico e reduzir riscos de negócio e implementação.
- Atuação em Tarefas (Visão de Engenharia de Software):
    - Tarefa: Análise de Trade-offs Arquiteturais para a Solução de vídeos interativos com hotspots.
        - O estudo técnico envolve uma profunda análise de engenharia de sistemas, considerando custos, escalabilidade e manutenibilidade. Minha tarefa foi fornecer à liderança um relatório técnico para uma decisão informada.
        - Execução
            1. Análise de Requisitos Não-Funcionais: Foi definido os critérios de avaliação: latência, escalabilidade , custo por espectador/hora e complexidade de manutenção.
            2. Relatório Técnico e Recomendação: Nele, apresentei a análise comparativa e recomendei a abordagem, alinhando a solução técnica com os objetivos de negócio (velocidade de lançamento), com um plano para reavaliar a estratégia se a escala crescesse exponencialmente.

4. Implementação de Lógicas para Descontos (Período: 04/10 a 06/11)

- Detalhamento da Execução: Foco total em refatoração segura e melhoria da qualidade de um módulo crítico e legado.
- Atuação em Tarefas:
    - Tarefa: Garantia de Qualidade e Prevenção de Regressão em Código Legado Crítico.
        - Modificar um código complexo em um sistema legado e monolítico, sem testes e com alto impacto no negócio (cálculo de preços), sendo uma tarefa de risco, a prioridade absoluta foi garantir que o comportamento existente não fosse quebrado.
        - Execução Baseada:
            1. Engenharia Reversa e Testes de Caracterização: Antes de alterar uma única linha, apliquei a técnica de Testes de Caracterização. Criei uma suíte de testes que não validava o que o código deveria fazer, mas o que ele realmente fazia, incluindo seus comportamentos inesperados. Isso criou uma rede de segurança que "caracterizou" o comportamento do sistema.
            2. Refatoração Baseada em Padrões: Com a segurança dos testes, liderei a refatoração. O código original era uma violação do Princípio Aberto/Fechado (para adicionar um novo desconto, era preciso modificar a função existente). Implementamos o Strategy Pattern, onde cada lógica de desconto se tornou uma classe separada com uma interface comum. A função principal agora apenas selecionava a "estratégia" correta, sem precisar conhecer seus detalhes.
            3. Manutenibilidade: A nova arquitetura tornou-se aberta para extensão (bastava criar uma nova classe de estratégia) e fechada para modificação, garantindo a manutenibilidade e a segurança para futuras alterações.

        - Tarefa ainda não finalizada: Necessário a criação de mais testes, a validação da tarefa em conjunto com o time de qualidade e a realização do deploy.

5. Implementação de Funcionalidade para Imagens no Website (Período: 25/09 a 15/10)
    - Detalhamento da Execução: Entregue de forma incremental, com foco inicial na funcionalidade e posterior na otimização de performance.
    - Atuação:
        - Tarefa: Otimização de Performance e Métricas de Core Web Vitals (CWV).
            - Minha atuação como engenheiro foi focada em garantir que a funcionalidade não degradasse as métricas de performance (LCP, CLS), que impactam diretamente a experiência do usuário e o ranking no Google.
            - Execução Baseada em Engenharia de Software:
                1. Análise e Diagnóstico: Utilizei ferramentas de profiling (Lighthouse, Chrome DevTools) para medir o impacto da implementação inicial. O diagnóstico foi claro: o carregamento de múltiplas imagens de alta resolução estava atrasando o LCP (Largest Contentful Paint) e o carregamento tardio causava CLS (Cumulative Layout Shift).
                2. Otimização Estrutural: Apliquei uma abordagem multifacetada:
                    - Usei o componente <Image> do Next.js, que automaticamente serve imagens em formatos modernos (WebP) e tamanhos otimizados.
                    - Adicionei a prop priority na primeira imagem do carrossel. Isso é uma diretiva de engenharia que sinaliza ao navegador para priorizar o download deste recurso, melhorando diretamente o LCP.
                    - Para as demais imagens, implementei lazy loading, fazendo com que fossem carregadas apenas quando se aproximasse do viewport.
                    - Para resolver o CLS, especifiquei as dimensões exatas de cada imagem, para que o navegador pudesse reservar o espaço em tela antes do download ser concluído, evitando o "salto" no layout.
                3. Validação: Após as otimizações, rodei novamente as ferramentas de profiling para validar que as métricas de CWV estavam dentro dos limites recomendados, tratando a performance como um requisito funcional do projeto.
                4. Publicação de nova versão: A versão foi publicada com as alterações, este deploy ocorreu no período da noite por conta dos riscos de serem realizados durante o dia.
        - Testes unitários realizados: Através dos testes unitários, na tabela 2, são evidenciados os critérios de assertividade da funcionalidade para garantir a integridade e a qualidade em ambiente local de desenvolvimento. Estes mesmos testes são executados antes do deploy da aplicação.

Tabela 2 - Testes unitários na Implementação de Funcionalidade para Imagens no Website

Objetivo	Verificação(Assert)
Validar a renderização inicial do componente.	Verificar se o número de elementos de imagem renderizados no DOM é igual ao tamanho do array passado na prop 'images'.
Testar a interatividade do botão 'próximo'.	Simular um clique no botão de avançar e verificar se a função de callback 'onNextClick' foi chamada exatamente uma vez.
Testar a interatividade do botão 'anterior'.	Simular um clique no botão de voltar e verificar se a função de callback 'onPrevClick' foi chamada exatamente uma vez.
Validar o estado desabilitado do botão 'anterior' no início.	Renderizar o componente no estado inicial e verificar se o botão 'anterior' possui o atributo 'disabled'.
Validar o estado desabilitado do botão 'próximo' no final.	Simular a navegação até o último slide e verificar se o botão 'próximo' possui o atributo 'disabled'.
Testar a sincronização da UI dos indicadores de paginação.	Verificar se o indicador (dot) correspondente ao slide ativo possui uma classe CSS de 'active' e os outros não.
Validar a renderização condicional baseada em props.	Renderizar o componente com a prop 'showArrows' como 'false' e verificar se os botões de navegação não existem no DOM.

Fonte: Do próprio autor

---

6. Atualização da versão do React Native (Período: 25/09 a 15/10)
Uma atividade de alto risco focada em gerenciamento de débito técnico e configuração de ambiente.

- Atuação em Tarefas:
    - Gerenciamento de Dependências Nativas Conflitantes e do Processo de Build.
        - A complexidade de uma atualização do React Native está no código JavaScript, e também no ecossistema nativo subjacente. É uma tarefa de Engenharia de Configuração dos componentes nativos, atualização

de bibliotecas do React-Native e correção de builds. Na tabela 3, estão as alterações após a migração da versão anterior para a nova atualização da versão 0.81.

- Execução:
    1. Análise de Risco e Planejamento Estratégico:
        - Análise de Breaking Changes: Antes de iniciar, utilizei a ferramenta oficial react-native-upgrade-helper. Ela fornece um diff detalhado entre as versões, mostrando todas as alterações necessárias em arquivos de template nativos e de configuração, o que foi crucial para estimar o esforço e os pontos de risco.
        - Mapeamento de Dependências: Criei um inventário de todas as bibliotecas de terceiros, verificando seus changelogs e issues no GitHub para confirmar a compatibilidade com a nova versão do React Native.
        - Plano de Execução: Defini um plano em uma branch Git isolada (feature/upgrade-rn-0.xx), que incluía um plano de rollback claro: a qualquer momento, se um impedimento intransponível fosse encontrado, a branch seria descartada, garantindo zero impacto na main branch
    2. Gerenciamento de Build Nativo: A atualização quebrou as configurações do Gradle (Android) e do CocoaPods (iOS). Meu trabalho foi mergulhar nesses ecossistemas nativos, entender as breaking changes e aplicar as correções manualmente, o que exigiu conhecimento de android nativo.
    3. Solução de Conflitos de Dependências: O problema mais crítico foi uma biblioteca de SDK com a Google que dependia de uma versão antiga de uma biblioteca nativa do Android, enquanto a nova versão do React Native exigia uma mais recente. A solução de engenharia foi usar as ferramentas do react-native para realizar uma atualização pontual, através de um patch, e então aplicar esse ajuste no código da biblioteca da Google (usando patch-package) para torná-la compatível.
    4. Validação: Após os ajustes, garantindo que o processo automatizado estivesse validado antes de mesclar as mudanças para a branch principal. Isso garantiu a reprodutibilidade do build para toda a equipe e passagem para as próximas etapas de testes com o analista de qualidade.
    5. Testes unitários: Garantindo a qualidade de entrega, estão mapeados na tabela 4, os testes unitários que foram necessários adicionar após a migração da versão do react-native.

Tabela 3 - Alterações após migração da versão do React-native

Ponto de Alteração	Descrição Técnica da Mudança	Solução
Atualização do Android Gradle	A nova versão do React Native exige uma versão mais recenteo Gradle. Isso implicou em atualizar as dependências no build.gradle do projeto e a URL de distribuição no gradle-wrapper.properties.	A nova versão do AGP introduziu breaking changes na sintaxe do build.gradle, especialmente na forma como as buildFeatures e o namespace são declarados. Solução: Foi necessário a atualização do gradle
Migração para TurboModules	Nossos módulos nativos customizados (escritos em Java) seguiam o padrão antigo (ReactContextBaseJavaModule). A Nova Arquitetura exige a migração para TurboModules.	Exigiu a reescrita de um dos módulos nativos. O código Java teve que ser adaptado para implementar as interfaces geradas pela especificação. Solução: Abordei a migração do módulo, garantindo o build do projeto após a migração do novo TurboModule.
Requisitos do Ecossistema iOS (Xcode)	A versão 0.81 passou a exigir uma versão mais recente do Xcode (ex: Xcode 16) e do iOS SDK (ex: iOS 17).	Isso representou um desafio de infraestrutura. Foi necessário atualizar o ambiente de desenvolvimento de todos os membros da equipe.
Depreciação de APIs Core	A nova versão removeu alguns componentes e APIs do core do React Native, movendo-os para pacotes da comunidade (ex: react-native-community).	instalação da nova dependência (@react-native-clipboard/clipboard), garantindo que nenhuma ocorrência fosse esquecida.

Fonte: Do próprio autor

Tabela 4 - Testes unitários na Atualização da versão do React Native

Tipo de Teste	Ferramentas/Método	Verificação(Assert)
Smoke Test (Manual)	Build de Teste (APK/IPA)	A aplicação abre sem crashar após a instalação limpa em um dispositivo físico com Android 14 e iOS 17.
Regressão de Módulos Nativos	Teste Manual Focado	Verificar se a funcionalidade do módulo de integração com o SDK do Google (agora como TurboModule) continua operando corretamente.
Regressão de Performance	Flipper, Perfetto, Xcode Instruments	Medir o tempo de inicialização (TTO) e o uso de memória em repouso. Comparar os resultados com a baseline da versão 0.76."

Fonte: Do próprio autor

Referências

[1] Schwaber, K., & Sutherland, J. (2020). The Scrum Guide.
[2] Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach.
[3] Beck, K., et al. (2001). Manifesto for Agile Software Development.
[4] Newman, S. (2015). Building Microservices.
[5] Martin, R. C. (2008). Clean Code.
[6] Frost, B. (2016). Atomic Design.
[7] Ries, E. (2011). The Lean Startup.

Análise do período de estágio
Dificuldades encontradas no período:

Conciliação de Processos Burocráticos Distintos: A principal dificuldade foi a necessidade de alinhar os processos administrativos de duas grandes instituições com fluxos e processos independentes: o departamento de RH da RDsaúde e a UNIPAMPA. Cada instituição possui seu próprio processo e jurisprudência, a sincronização da documentação exigiu uma mediação constante para garantir que os prazos e exigências de ambos os lados fossem cumpridos.
Adaptação do Vínculo Empregatício para o de Estágio: Por já possuir um vínculo de trabalho efetivo, a transição ou adaptação para o modelo de estágio não era um procedimento padrão para a empresa. Isso envolveu consultas ao departamento jurídico e de RH da RDsaúde para entender como formalizar o estágio sem conflitar com o contrato de trabalho existente, gerando uma camada extra de complexidade administrativa que um candidato externo não enfrentaria.
Elaboração de um Plano de Atividades Acadêmico Profissional Coerente: O desafio foi criar um Plano de Atividades que atendesse simultaneamente a dois critérios:

1. Requisitos da UNIPAMPA: Precisava ser academicamente robusto, com objetivos claros de aprendizado e alinhado ao projeto pedagógico do curso.
2. Realidade Profissional: Precisava refletir com precisão as minhas responsabilidades e projetos já em andamento na RDsaúde, que são dinâmicos e guiados pelas necessidades de negócio da empresa (Sprints, demandas de produto, etc.). A tarefa foi traduzir as atividades de um profissional de mercado para uma linguagem e estrutura que fossem formalmente aceitas pela universidade.

Definição e Alinhamento dos Papéis de Supervisão: Foi necessário alinhar as expectativas entre o meu gestor direto na RDsaúde (supervisor de estágio na empresa) e o professor orientador designado pela UNIPAMPA.

Complexidade de Ambientes Nativos e Multiplataforma: A atualização da versão do React Native expôs a complexidade inerente à gestão de um ecossistema que depende diretamente das configurações nativas de Android (Gradle) e iOS (CocoaPods). Enfrentar *breaking changes* e conflitos de dependências exigiu um aprofundamento em conhecimentos de desenvolvimento nativo que vão além do escopo do JavaScript.

Trabalho com Código Legado e Débito Técnico: A tarefa de refatoração do módulo de descontos apresentou o desafio de modificar um sistema monolítico, crítico para o negócio, que carecia de testes automatizados e possuía regras de negócio não documentadas. O risco de introduzir regressões era altíssimo, exigindo uma abordagem metódica e cautelosa com testes de caracterização antes de qualquer alteração.

Otimização de Performance e Métricas Web: A implementação de funcionalidades, como o carrossel de imagens, revelou que a entrega de valor não se resume à funcionalidade em si, mas também à sua performance. A necessidade de otimizar os Core Web Vitals (LCP, CLS) exigiu um conhecimento aprofundado de como o navegador renderiza as páginas e das ferramentas de profiling (Lighthouse, DevTools) para diagnosticar e solucionar gargalos de performance.

unipampa
Universidade Federal do Pampa

Competências e conhecimentos adquiridos no período:

Arquitetura de Software e Design Patterns: Aplicação prática de padrões de arquitetura (Microsserviços, API Gateway) e de design (Strategy Pattern) para construir sistemas desacoplados, manutenível e extensíveis, seguindo os princípios SOLID e de Código Limpo.

Engenharia de Qualidade de Software (QA): Desenvolvimento de uma mentalidade focada em qualidade, com a implementação de suítes de testes (unitários, caracterização) como rede de segurança para refatorações e a utilização de testes para garantir a robustez de novos componentes.

Otimização de Performance Web e Móvel: Conhecimento aprofundado em diagnóstico e otimização de performance, utilizando ferramentas como Lighthouse e Chrome DevTools para melhorar métricas de Core Web Vitals (LCP, CLS) e aplicando técnicas de otimização em React (memoização, hooks useRef/useCallback) para garantir a fluidez da interface.

Gestão de Configuração e Build Multiplataforma: Competência para gerenciar e depurar os sistemas de build nativos de Android (Gradle) e iOS (CocoaPods) no contexto de um projeto React Native, incluindo a resolução de conflitos complexos de dependências (dependency hell) e a aplicação de patches.

Liderança Técnica e Metodologias Ágeis: Habilidade de liderar tecnicamente a execução de tarefas complexas, facilitar discussões de arquitetura, mentorar a equipe em boas práticas e utilizar ativamente as cerimônias do Scrum para garantir a transparência, a colaboração e a remoção de impedimentos.

Análise de Viabilidade Técnica e de Negócio: Capacidade de traduzir requisitos de negócio em investigações técnicas (Spikes), construir Provas de Conceito (PoCs) e elaborar relatórios técnicos com análises comparativas (trade-offs) e de custos para embasar decisões estratégicas da liderança.

Justificativas para o não cumprimento de atividades no período:

Este relatório, com prazo de entrega originalmente previsto para o dia 13 de outubro, está sendo submetido no dia 16 de outubro.

O atraso de três dias se deve à alta intensidade das demandas de projeto na semana que antecedeu o prazo. Especificamente, o período coincidiu com o ciclo final de uma Sprint que envolveu **deploys** (processos de implantação de software) em ambiente de produção. Essas atividades, por sua natureza crítica, são frequentemente realizadas fora do horário comercial, incluindo o período noturno, para minimizar o impacto sobre os usuários. Essa dinâmica exigiu

uma dedicação de tempo e energia que se estendeu para além da jornada de trabalho regular, impactando a disponibilidade para a consolidação e redação detalhada deste relatório dentro do prazo original.

Ciente de que a natureza do meu trabalho pode levar a picos de demanda semelhantes no futuro, especialmente em períodos de fechamento de Sprint e deploys, gostaria de propor uma alternativa para os próximos relatórios. Os prazos podem ser mantidos conforme o cronograma original. Nesses casos, compreendo que a responsabilidade pela entrega recairá sobre minha capacidade de gerenciar o tempo, o que pode envolver a utilização de horas durante os finais de semana ou no período noturno para a contagem das horas e elaboração dos relatórios.

Bagé, 16 de Outubro de 2025.

gov.br Documento assinado digitalmente
LEONARDO BIDESE DE PINHO
Data: 17/10/2025 06:50:52-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Leonardo Bidese de Pinho – SIAPE 1651472
Orientador na UNIPAMPA

gov.br Documento assinado digitalmente
MICHEL ALMEIDA DA SILVA
Data: 17/10/2025 07:45:40-0300
Verifique em https://validar.iti.gov.br

_____

Michel Almeida da Silva
Discente – Matrícula 1801570095