



Mooras - Um Aplicativo Para Gestão e Logística de Vendedores Autônomos

Nome Daniel Ferreira Costa

Curso Desenvolvimento Full Stack

Orientador(a) Raphael Gomide

Data 08/04/2021

Sumário

1. CANVAS do Projeto Aplicado	4
1.1. Desafio	4
1.1.1. Análise de Contexto	4
1.1.1.1. Matriz CSD	5
1.1.1.2. Matriz POEMS	5
1.1.2. Personas	5
1.1.3. Benefícios e Justificativas	6
1.1.3.1. Business Design Blueprint	6
1.1.3.2. Proposta de Valor	7
1.1.4. Hipóteses	8
1.1.4.1. Brainstorm	8
1.1.4.2. Priorização de Ideias	8
1.2. Solução	9
1.2.1. Objetivo SMART	9
1.2.2. Premissas e Restrições	9
1.2.3. Backlog de Produto	10
1.2.3.1. Concept Backlog	10
1.2.3.2. Product Backlog	10
2. Área de Experimentação	11
2.1. Sprint 1	11
2.1.1. Solução	11
2.1.1.1. Evidencias do Planejamento	11
2.1.1.2. Evidência da execução de cada requisito	11
2.1.1.3. Evidência de solução	12
2.1.2. Lições Aprendidas	17
2.2. Sprint 2	17
2.2.1. Solução	17
2.2.1.1. Evidencias do Planejamento	17
2.2.1.2. Evidência da execução de cada requisito	18
2.2.1.3. Evidência de solução	18
2.2.2. Lições Aprendidas	19
2.3. Sprint 3	20
2.3.1. Solução	20
2.3.1.1. Evidencias do Planejamento	20
2.3.1.2. Evidência da execução de cada requisito	20

2.3.1.3. Evidência de solução	22
2.3.2. Lições Aprendidas	23
2.4. Sprint 4	23
2.4.1. Solução	24
2.4.1.1. Evidências do Planejamento	24
2.4.1.2. Evidência da execução de cada requisito	24
2.4.1.3. Evidência de solução	25
2.3.2. Lições Aprendidas	29
2.5. Sprint 5	29
2.5.1. Solução	29
2.5.1.1. Evidências do Planejamento	29
2.5.1.2. Evidência da execução de cada requisito	30
2.5.1.3. Evidência de solução	31
2.5.2. Lições Aprendidas	37
2.6. Sprint 6	37
2.6.1. Solução	37
2.6.1.1. Evidências do Planejamento	37
2.6.1.2. Evidência da execução de cada requisito	38
2.6.1.3. Evidência de solução	38
2.6.2. Lições Aprendidas	40
2.7. Sprint 7	41
2.7.1. Solução	41
2.7.1.1. Evidências do Planejamento	41
2.7.1.2. Evidência da execução de cada requisito	41
2.7.1.3. Evidência de solução	42
2.7.2. Lições Aprendidas	43
3. Considerações Finais	44
3.1 Resultados Finais	44
3.2 Contribuições	44
3.3 Próximos passos	44

1. CANVAS do Projeto Aplicado

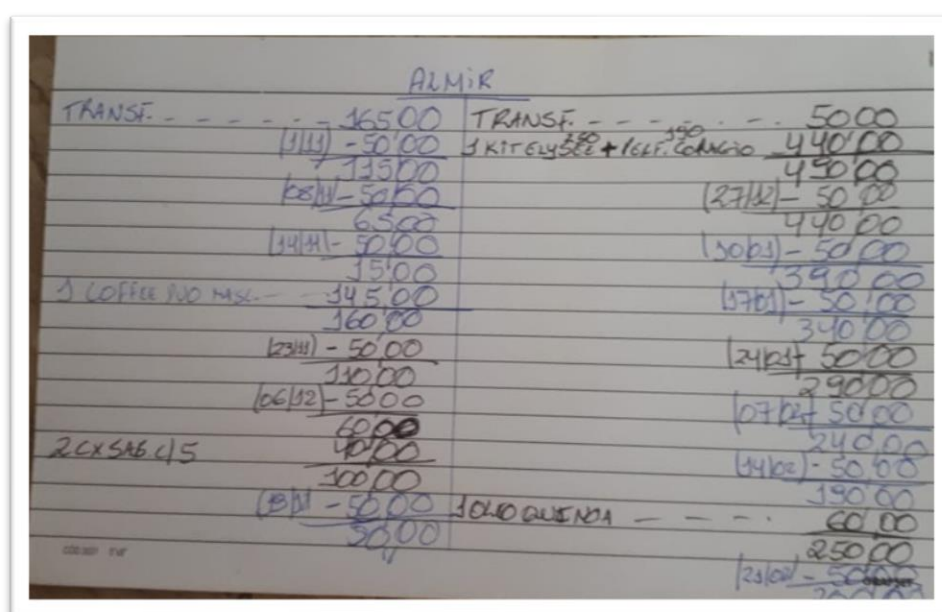
Este documento consiste no relatório de desenvolvimento de uma solução trazendo problemas do mundo real. Neste primeiro capítulo será realizado um estudo do desafio proposto, para então definir a melhor solução, e a partir daí seguir para as demais etapas.

1.1. Desafio

Neste subcapítulo será definido o desafio, bem como explicado suas motivações.

1.1.1. Análise de Contexto

Ainda nos dias de hoje, é comum observar revendedores de cosméticos gerenciando suas vendas por meio de cadernetas, a exemplo da Figura 1. Este hábito possui uma logística rápida e fácil, entretanto, apenas para os casos em que o número de clientes é pequeno.



ALMIR	
TRANSF. - - - - - 165,00	TRANSF. - - - - - 50,00
(11/11) - 50,00	1 KIT ELYSÉE + 1 KIT. GORGEO 440,00
135,00	450,00
(18/11) - 50,00	(27/11) - 50,00
65,00	400,00
(14/11) - 50,00	(30/11) - 50,00
15,00	350,00
1 COFFEE NO MXL - - - 345,00	(17/11) - 50,00
360,00	300,00
(23/11) - 50,00	(24/11) - 50,00
310,00	250,00
(06/12) - 50,00	(07/12) - 50,00
60,00	200,00
2 CX SAB 1/5 40,00	(14/12) - 50,00
300,00	150,00
(19/12) - 50,00	100,00
50,00	250,00
	(23/12) - 50,00
	200,00

Figura 1 – Ficha de Cliente – Autoria Própria

Quando se trata de um grande número de clientes, consequentemente um número maior de vendas, gerir se torna uma tarefa árdua. O mais simples dos ofícios que é encontrar a ficha do cliente, as vezes pode encontrar dificuldades, pois, encontrar uma determinada ficha no meio de tantas outras durante uma venda pode demorar. Uma vez que, em um caderno nem sempre é possível anotar suas vendas em uma ordem específica (alfabética por exemplo).

Acresce a isso, o caderno traz dificuldades administrativas, como criação de relatórios, exemplo: descobrir a soma total que todos os clientes lhe devem, quais foram os produtos mais vendidos, quanto de estoque existe de cada produto. Todos esses valores, em um caderno, teriam que ser calculados manualmente de e de forma individual.

Todo esse trabalho manual, extrai tempo do revendedor de cosméticos, que poderia ser redirecionado para as próprias vendas em si ao invés do gerenciamento das vendas.

1.1.1.1. Matriz CSD

Para explicar melhor o problema, foi elaborado uma *Matriz de Certezas, Suposições e Dúvidas* (CSD), onde podemos ver com mais clareza essa informação:

	Matriz CSD		
	Certezas	Suposições	Dúvidas
Atores	<ul style="list-style-type: none"> Pessoas que vendem cosméticos e querem ter maior controle do seu negócio; Pessoas que vendem cosméticos e desejam reduzir o tempo de gestão de vendas. 	<ul style="list-style-type: none"> Pessoas desejam abandonar os blocos de anotações; Nem todas as pessoas sabem utilizar aplicativos de celular. 	<ul style="list-style-type: none"> Que tipo de aplicativo o cliente estaria disposto a usar?
Cenários	<ul style="list-style-type: none"> A maioria das pessoas possuem celular; Existem sites e aplicativos com funcionalidades semelhantes. 	<ul style="list-style-type: none"> Pessoas usariam um aplicativo que resolve todo ou parte do problema. 	<ul style="list-style-type: none"> O excesso de informação disponível prejudica a usabilidade?
Regras	<ul style="list-style-type: none"> Clientes podem pagar à vista ou parcelado. 		

Matriz CSD – Autoria Própria

1.1.1.2. Matriz POEMS

Uma das principais características de revendedores de cosméticos é que não possuem uma loja física, nem um lugar específico que possam vender. Sendo assim, qualquer ambiente se torna alvo para uma possível venda. Para identificar melhor situações como esta, também foi desenvolvido uma matriz *POEMS* (Pessoas, Objetos, Ambiente, Mensagem e Serviços).

Matriz POEMS				
Pessoas	Objetos	Ambientes	Mensagem	Serviços
Vendedor	Produtos	Diversos	Mensagens que visam detalhar os benefícios dos produtos oferecidos	Venda de produto
Cliente	Recursos monetários	Diversos	Mensagens que especificadoras do produto buscado	

Matriz POEMS – Autoria Própria

1.1.2. Personas

Com o objetivo de criar maior empatia com aquela pessoa que mais sofre do problema e mais estaria adepto a experimentar uma nova solução, foi criado o perfil da persona, para representar os diferentes tipos de usuário dentro de um alvo demográfico, atitude e/ou comportamento definido que pode ser identificado como sendo um cliente fictício e ideal para a nossa possível solução.

Foi considerado para a identificação desse perfil as respostas obtidas na pesquisa de campo realizada para este projeto:

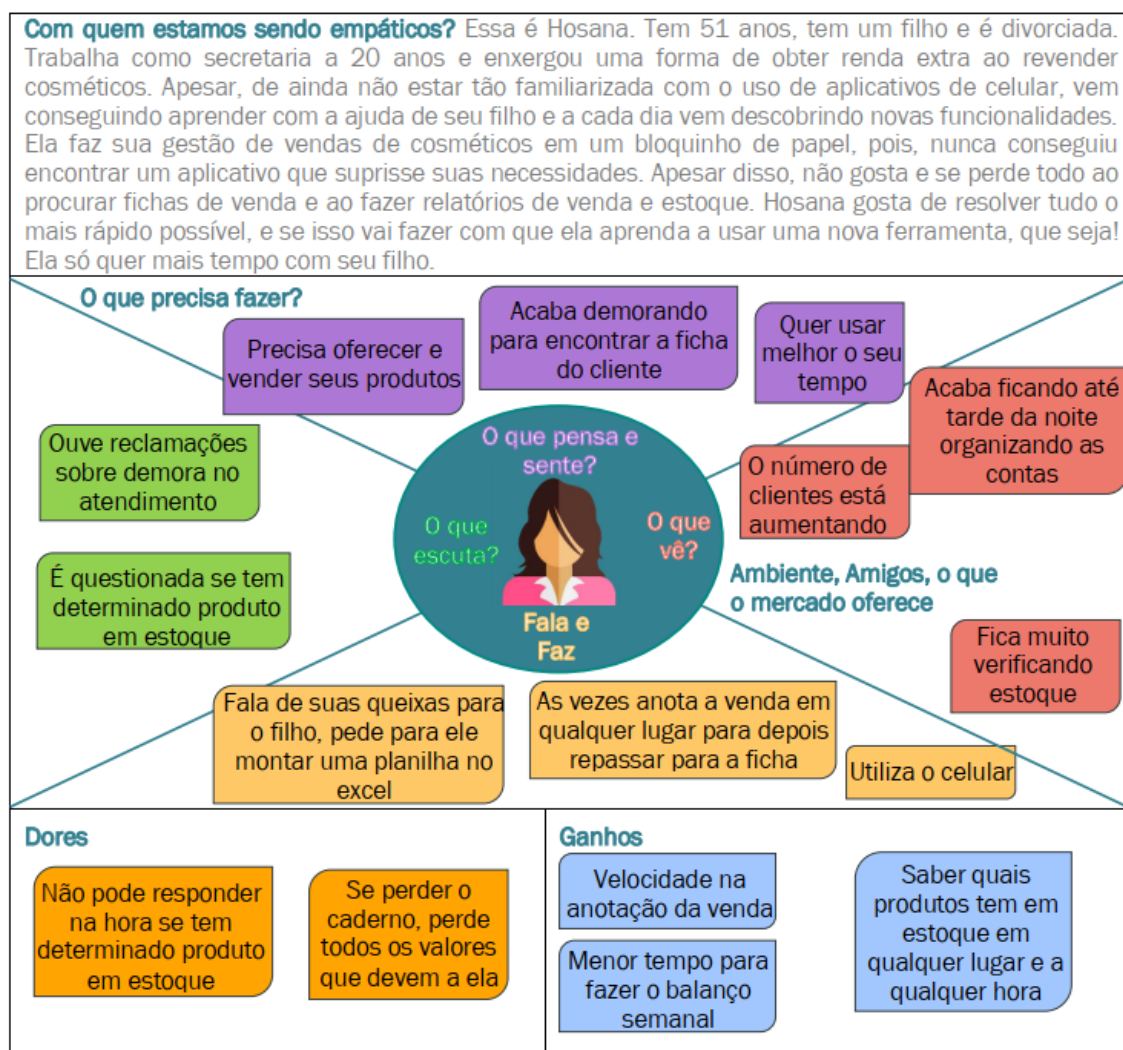


Gráfico da Persona – Autoria Própria

1.1.3. Benefícios e Justificativas

Encontrar uma solução para esse problema tem como consequência um aumento na produtividade e no tempo ‘ocioso’ das pessoas.

Quando se pode automatizar, otimizar e excluir tarefas que são feitas manualmente e exigem um grande esforço, motiva os vendedores a continuar suas vendas. E ainda sobra mais tempo para tarefas que não necessariamente tem relação com produtividade, mas com qualidade do uso do tempo.

Os subcapítulos a seguir tem como objetivo apresentar os benefícios que motivam o desenvolvimento do projeto.

1.1.3.1. Business Design Blueprint

Aqui podemos identificar por meio da Business Design Blueprint os passos realizados pelos usuários e todos os pontos desejáveis para a possível solução.

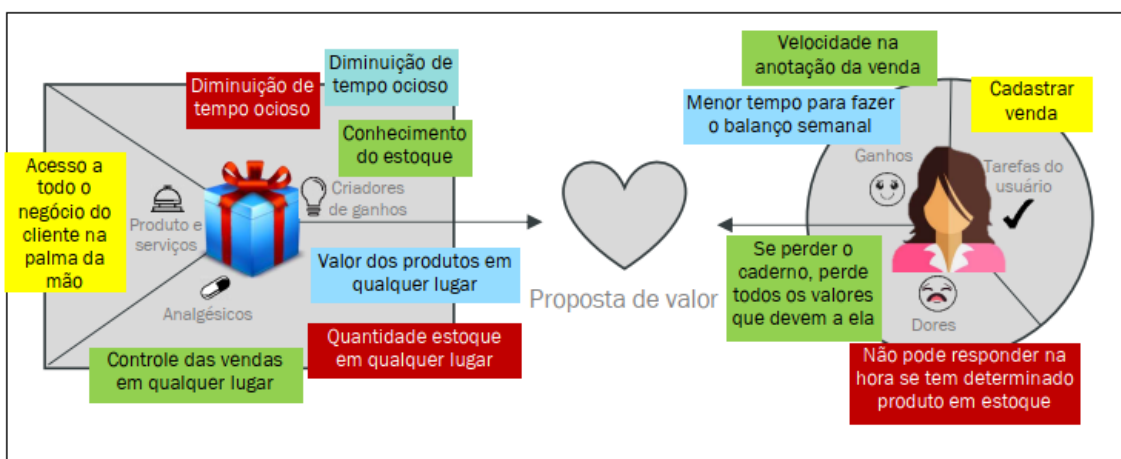
Aplicativo de celular para gerenciar vendas de produtos de cosméticos					
Ações do vendedor	Abrir o aplicativo	Escolher o cliente	Opção de cadastrar venda	Informar item, quantidade e valor	Finalizar cadastro da venda
Objetivos	Gerenciar suas vendas	Analisar determinado cliente	Cadastrar uma nova venda a um cliente	Detalhar a venda	Ter seus dados de venda atualizados
Atividades	Procura aplicativo e abre	Procura e filtra cliente	Escolhe, entre as opções para o cliente, cadastrar a venda	Filtra produto por nome, indica a quantidade e o valor	Finaliza ao clique do usuário
Mensagem	Gestão de vendas a um clique	Cada cliente na palma de sua mão	Gerencie seus clientes	Controle seu estoque em 1 segundo	Relatórios a um clique
Saída desejável	Estimular abandonar os cadernos de anotações	Acesso facilitado a ficha do cliente	Realizar cadastro		Ter uma experiência inovadora e incrível

Business Design Blueprint – Autoria Própria

Como identificado na matriz anterior, o acesso facilitado a ficha do cliente por meio de filtros e dados do produto pré-cadastrados facilitará em muito a vida do vendedor, uma vez que por ter os preços do produto no próprio aplicativo, o vendedor não necessariamente precisa carregar as revistas de preço.

1.1.3.2. Proposta de Valor

Uma vez em que se consegue olhar de perto as dores da persona, é possível agora pensar em algo que traga verdadeiro benefício. Com a ajuda do mapa de proposta de valor, foi identificado vários pontos para ajudar ainda mais a enfatizar o problema e então encontrar uma possível solução.



Mapa de Proposta de Valor – Autoria Própria

Como é possível ver na proposta de valor, as principais dores da persona são: Não pode responder na hora se tem determinado produto em estoque e a possibilidade de perder o caderno com todos os valores que devem a ela.

Para isso, os analgésicos pensados são: ter os valores dos produtos em qualquer lugar, ter a quantidade de produto em estoque em qualquer lugar e poder controlar as vendas em qualquer lugar. Em suma, ter todo o controle do negócio na palma de sua mão.

1.1.4. Hipóteses

Para explicar melhor as possibilidades, serão apresentadas as observações do problema, seguida de cada uma e suas hipóteses:

Observações	Hipóteses
Os vendedores gastam muito tempo buscando ficha dos clientes	Os vendedores gostariam de ter uma forma de filtrar a ficha do cliente
Os vendedores não tem real noção do volume do estoque	Os vendedores gostariam de uma forma automática ou semiautomática de controle de estoque
Os vendedores não tem de modo facilitado informações das vendas	Os vendedores gostariam de uma forma automática para criar relatórios
Os vendedores, caso perca o caderno com as fichas, perde todos os seus dados	Os vendedores gostariam que os dados fossem salvos na nuvem

Matriz de hipóteses – Autoria Própria

1.1.4.1. Brainstorm

Após todas essas análises, realizou-se um brainstorm, que deu origem à uma lista com possíveis soluções para esse problema:

- Utilizar ferramentas já existentes que não são focadas em vendedores de cosméticos;
- Aplicativo mobile para os vendedores gerenciar seu negócio;
- Aplicação web para os vendedores gerenciar seu negócio;
- Organizar as listas em fichas por ordem alfabética ao invés de caderno e atualizar dados como estoque sempre ao final do dia.

1.1.4.2. Priorização de Ideias

Para escolher qual das ideias listadas anteriormente, foi desenvolvido uma matriz, utilizando como critério os pontos de BASICO (benefícios, abrangência, satisfação, investimento, cliente e operação):

Solução para gerenciamento de filas							
IDEIAS	B	A	S	I	C	O	TOTAL
Utilizar ferramentas já existentes que não são focadas em vendedores de cosméticos	4	4	3	5	5	4	25
<i>Aplicativo mobile para os vendedores gerenciar seu negócio</i>	5	5	5	3	5	4	27
Aplicação web para os vendedores gerenciar seu negócio	5	5	4	3	5	4	26
Organizar as listas em fichas por ordem alfabética ao invés de caderno e atualizar dados como estoque sempre ao final do dia	2	3	2	5	4	3	19

Matriz de priorização de ideias (BASICO) – Autoria Própria

A solução *Aplicativo mobile para os vendedores gerenciar seu negócio* obteve 27 pontos utilizando esse método, por tanto, essa é a solução escolhida para continuar os próximos passos desse projeto.

1.2. Solução

A solução escolhida para seguir para as próximas fases é a criação de uma aplicação mobile para que a persona administre seu negócio, desde o gerenciamento de clientes, a vendas e produtos.

1.2.1. Objetivo SMART

Para identificar melhor os pontos a serem trabalhados pela solução, foram listados os objetivos do mesmo seguindo a linha SMART (*Specific, Mensurable, Attainable, Relevant, Time based*). São eles:

- Mockar solução para facilitar desenvolvimento dentro de 2 semanas;
- Prototipar solução utilizando desenhos de alta fidelidade dentro de 2 meses;
- Estruturar um mock API para implementação de protótipo de alta fidelidade em 4 meses;
- Desenvolver MVP da solução em 8 meses.

1.2.2. Premissas e Restrições

Tendo agora a persona Hosana desenhada, com a solução já definida e os objetivos do projeto estruturado, é possível identificar outros pontos importantes para o andamento desse trabalho.

1.2.2.1. Premissas

Para começar, abaixo há uma lista de premissas que devem ser seguidas para esse projeto:

- Deve se considerar 5 horas semanais para a execução desse projeto;
- Ter o máximo de acessibilidade deve ser sempre prioridade da aplicação;
- A aplicação deve ter o mínimo de telas possíveis, para facilitar o uso.

1.2.2.2. Restrições

- Um MVP precisa ser entregue até outubro;
- Será necessário ter armazenamento em nuvem para que não haja perda de dados em caso de troca de aparelho;
- A aplicação precisa ser um aplicativo mobile.

1.2.2.2. Matriz de Risco

Assim como em todo projeto, há algumas tarefas que merecem atenção, para que não se tornem problemas futuros e impactem diretamente com o desenvolvimento e evolução da proposta. Para evitar algumas delas, foi pensado em planos de ação, que podem ser vistos a seguir:

Risco identificado	Impacto potencial	Ações preventivas	Ações corretivas
Atrasar entregas de sprint	Desenvolvimento do projeto	Manter organização dos horários para que não haja conflito	Usar horas extras no fim de semana para recuperar o tempo perdido
Vendedores não querem baixar o aplicativo	Valor do projeto	Utilizar uma PWA, para que a obrigatoriedade do download não exista	Campanhas explicando como baixar o app e adicioná-lo à tela inicial

Matriz de riscos – Autoria Própria

1.2.3. Backlog de Produto

Para início do desenvolvimento das sprints, será necessário primeiro a identificação dos passos a serem seguidos durante as experimentações do projeto, para isso, aqui vão as tarefas a serem desenvolvidas durante o projeto:

1.2.3.1. Concept Backlog

Na Figura 2, é possível observar o controle de desenvolvimento dos conceitos por meio do Trello.

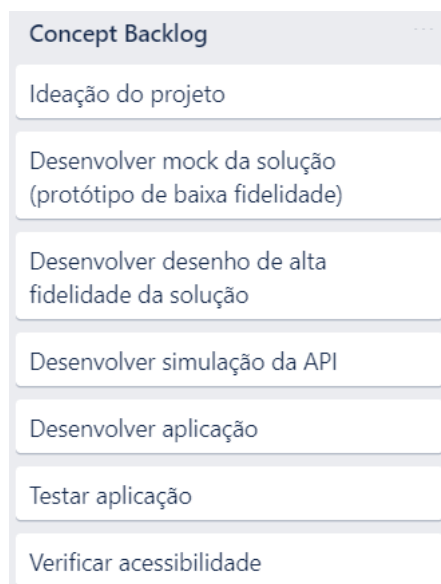


Figura 2 – Concept Backlog – Autoria Própria

1.2.3.2. Product Backlog

Na Figura 3, é possível observar o controle de desenvolvimento do produto por meio do Trello.

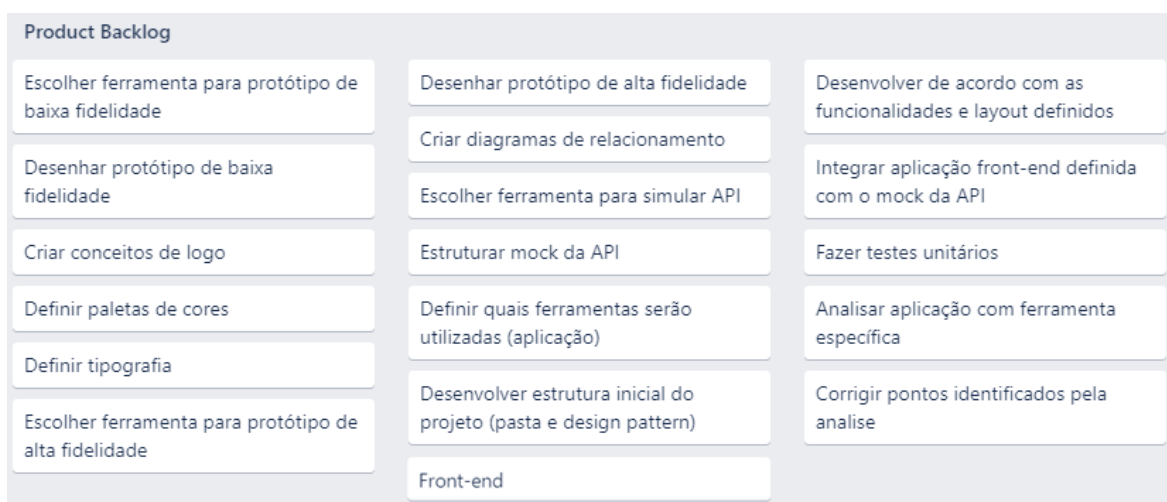


Figura 3 – Product Backlog – Autoria Própria

2. Área de Experimentação

Nessa seção se encontra a descrição de todas as sprints relacionadas ao projeto. Cada sprint sempre terá uma resolução para um *Product Backlog*, em grupos sempre resolverão um dos pontos do *Concept Backlog*.

2.1. Sprint 1

Nessa sprint, será desenvolvido os protótipos de baixa e alta fidelidade o projeto, já idealizando as idealizando as interações e UI do mesmo. Assim como, será definido a identidade visual do projeto.

2.1.1. Solução

A seguir, os passos e resultados as etapas para definição e design e identificação visual do projeto.

2.1.1.1. Evidencias do Planejamento

Para registro do que foi planejado nessa Sprint, segue a captura de tela da ferramenta Trello, Figura 4, onde se encontra na lista as tarefas realizadas nesta sprint.

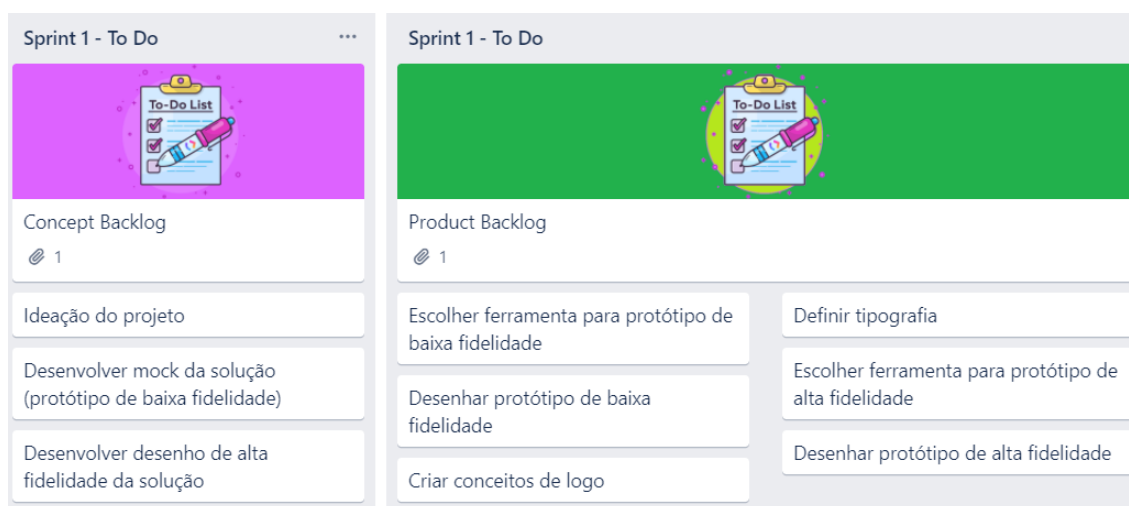


Figura 4 – Tarefas realizadas no sprint 1 – Autoria Própria

2.1.1.2. Evidência da execução de cada requisito

Durante o processo de desenvolvimento tanto do esboço das usabilidades e tela quanto da identidade visual, foram feitas algumas dessas versões antes de definir a que de fato vai ser utilizada. Algumas dessas versões foram guardadas para fins de registro do desenvolvimento.

Para esse esboço foi definido que o protótipo de baixa fidelidade seria melhor representado por meio de interações utilizando caneta e papel, fazendo desenhos livres de como cada tela e cada forma de interação foi pensada.

Inicialmente foram feitos esboços, Figura 5, do que poderia ser cada funcionalidade do aplicativo.

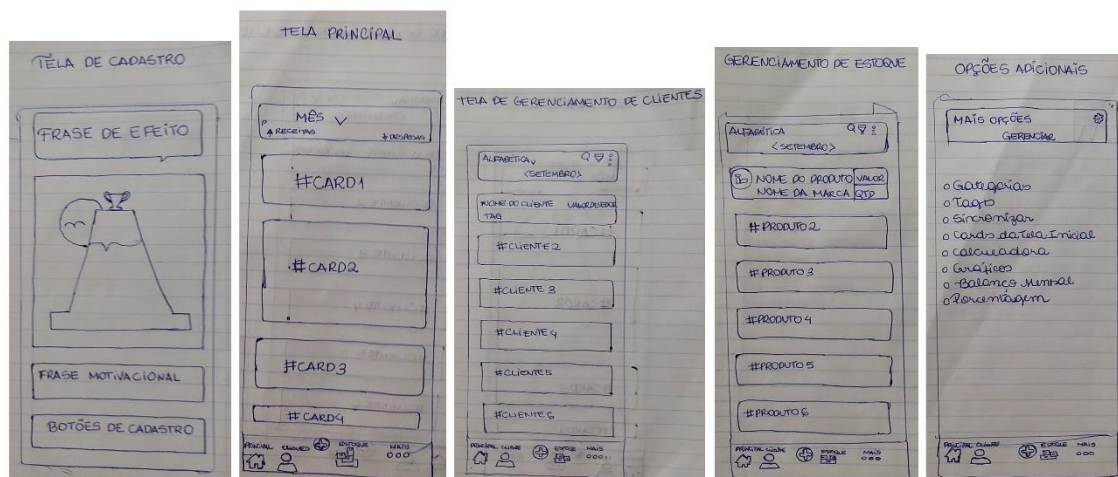


Figura 5 – Protótipo de baixa fidelidade das telas – Autoria Própria

Além disso, também foi idealizado uma primeira versão do logo do aplicativo, Figura 6. Já em relação a tipografia e paleta de cores, os testes forem feitos no protótipo de alta fidelidade.

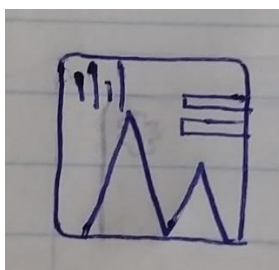


Figura 5 – Protótipo do logo do aplicativo – Autoria Própria

2.1.1.3. Evidência de solução

Neste subcapítulo é apresentado a solução para os 'To Do' das tarefas apresentadas na lista do Trello.

- **Protótipo de baixa fidelidade**

O protótipo de baixa fidelidade, foi executado por meio de caneta e papel, e apresentado no subcapítulo 2.1.1.2. como esboço e evidencia do planejamento e execução do projeto.

- **Marca e identidade visual**

Antes de apresentar a logo, é importante entender alguns pontos que serviram de inspiração para a criação da mesma.

- Visão: ser a plataforma de controle finanças com maior número de vendedores de cosméticos como clientes em todo o mundo.
- Missão: oferecer aos clientes meios para atingir a tranquilidade financeira em seu negócio.
- Valores: qualidade, segurança, valorização das pessoas, trabalho em equipe de forma colaborativa e ambiente descontraído.

O nome do aplicativo foi escolhido como 'Mooras' – que tem um significado especial por meio da aglutinação da frase 'Mobilidade nas Alturas'.

Com base na visão, missão, valores e principalmente no nome do aplicativo foi desenvolvido a logo, Figura 6. Com adição de elementos representativos de ferramentas presentes, como gráficos, listas e outros.



Figura 6 – Logo do aplicativo Mooras – Autoria Própria

Para desenvolvimento do logo, paleta de cores e também a tipografia, foi utilizado a ferramenta de criação de identidade visão ‘Tailor Brands’.

Também utilizando como base o que usado no logo, a tipografia, bem como o seu uso, foi definida de acordo com a Figura 7.

Nome da fonte de texto	Roboto
Nome da fonte de ícone	Calibri
Título de página	Texto estilo negrito com tamanho 24 centralizado
Título de listas	Texto estilo regular com tamanho 24
Texto principal de listas	Texto estilo regular com tamanho 20
Botões	Texto negrito com tamanho 18 centralizado
Links	<u>Texto minúsculo com sublinhado e tamanho variável</u>

Figura 7 – Tipografia e descrição de uso – Autoria Própria

Baseado no que foi utilizado no logo, com auxílio da ferramenta ‘Colourco, foi pensado na paleta de cores que será utilizada para o sistema, conforme a Figura 8.

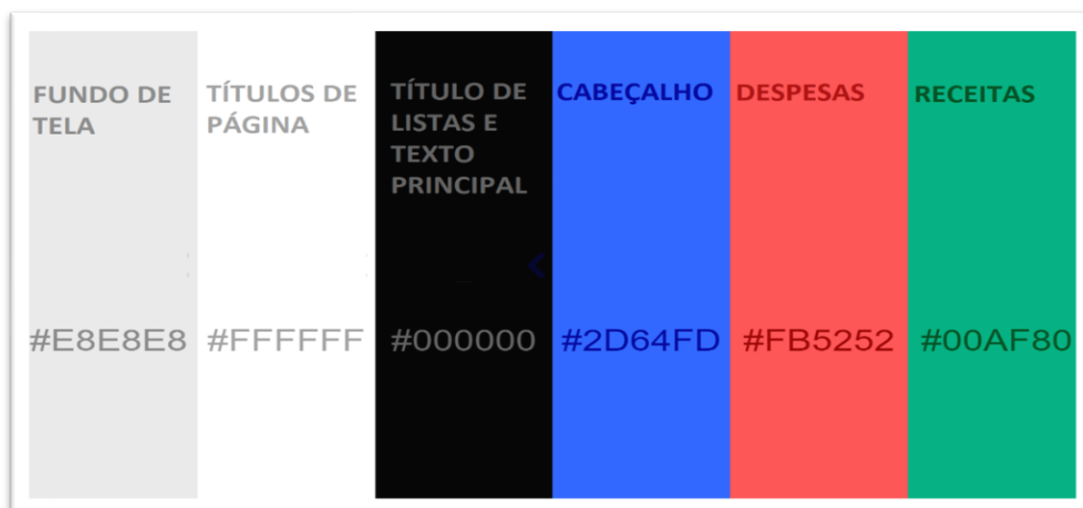


Figura 8 – Paleta de cores – Autoria Própria

- **Protótipo de alta fidelidade**

No protótipo de alta fidelidade foi desenvolvida as principais telas com auxílio da ferramenta 'Figma'. A cada tela será explicado detalhes sobre ela e as ferramentas presentes.



A Figura 9 demonstra a primeira tela após o usuário baixar o aplicativo, antes de fazer o login. Nesta tela é possível seguir dois caminhos:

- **CADASTRAR:** será encaminhado para uma tela, que permite, aos novos usuários que ainda não possui cadastro no Mooras possam se cadastrar (deve-se informar o nome completo, o e-mail e criar uma senha).
- **JÁ SOU CADASTRADO:** será encaminhado para uma tela, que permite usuários já cadastrados acessar as funcionalidades do aplicativo.

Após a realização do *login*, conforme a Figura 10, existem quatro telas que podem ser alternadas: tela principal; tela de gerenciamento de clientes; tela de controle de estoque; e a dela das demais operações. Além disso, existem um atalho em formato de '+' para adicionar uma nova venda.

Figura 9 – Tela de boas-vindas
– Autoria Própria



Figura 10 – Menu de telas –Autoria própria

A Figura 11 expõe a tela principal. Em cima, é possível observar um cabeçalho onde pode selecionar o mês de interesse e vê as principais informações sobre ele, como receitas e despesas. Logo abaixo do cabeçalho existem os *cards*, são cartões com todo tipo de dado, na figura pode observar que o primeiro cartão é um calendário com que marca datas significativas para o usuário, já o segundo cartão mostra um gráfico dos produtos mais vendidos naquele mês. A ideia dessa tela é que o usuário possa escolher quais *cards* devem aparecer na tela principal e a ordem em que eles devem aparecer, de acordo com sua preferência e relevância. Um outro exemplo de cartão é a listagem em ordem cronológica das vendas com informação do produto, valor e cliente.



Figura 11 – Tela principal –
Autoria Própria

Já na Figura 12, mostra a tela de gerenciamento de clientes e de gerenciamento de estoque, respectivamente. Nela é possível ordenar, buscar, filtrar, gerar relatórios, verificar detalhes sobre clientes/produtos, cadastrar novo cliente/produto, entre outros. Na tela de cliente por exemplo, ao clicar em um cliente, são informados dados como: nome, telefone, grupo, histórico de compras, histórico de pagamentos e dívida pendente.

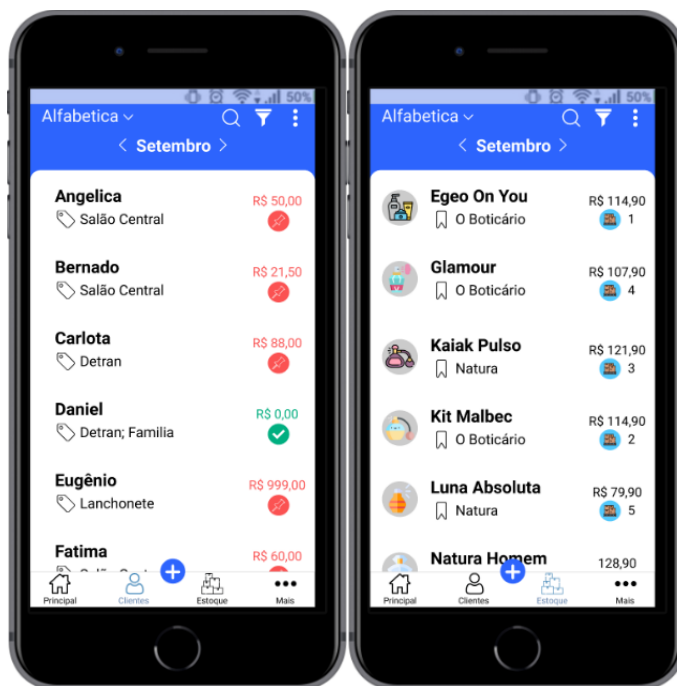


Figura 12 – Tela de gerenciamento de cliente e tela de gerenciamento de estoque –Autoria própria

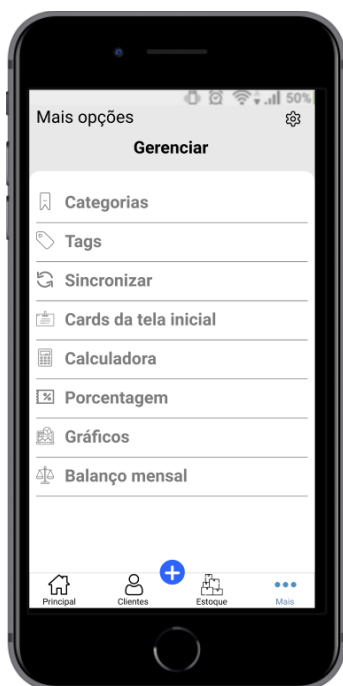


Figura 13 – Tela opções adicionais – Autoria Própria

Por fim, a Figura 13 representa a tela das demais opções (ou opções adicionais). Nela é possível achar ferramentas como calculadora e calculo de porcentagem de valores. Escolher quais cards vão aparecer na tela inicial. Criar grupos de clientes e de produtos. Consultar gráficos e balanços de valores.

Uma das funcionalidades mais importantes presente nessa tela é a de sincronizar. Onde é possível salvar os dados na nuvem. Uma vez que é um aplicativo que funciona como fichas de clientes e estoque, a perda desses dados pode causar um grave dano ao vendedor. O armazenamento em nuvem evita problemas ao perder o celular ou ter o mesmo furtado/roubado, evitando futuras dores de cabeça para os usuários (vendedores) e até mesmo aos clientes dos próprios usuários.

2.1.2. Lições Aprendidas

Dentro deste contexto, a sprint denota que os principais conteúdos colocados em prática fazem referência a disciplina de “Inovação e Design Thinking”, principalmente na produção em si do relatório apresentado e no que diz respeito a prototipação do aplicativo.

Destarte, concluiu-se que o processo de elaboração do protótipo de alta fidelidade apontou maior nível de complexidade, pois, inicialmente a prototipação havia sido elaborada por meio de ferramentas de edição de imagens, tais como o ‘Photoshop’ e ‘Paint’. Posteriormente, ao buscar por dispositivos mais específicos, fora escolhido o ‘Figma’ em razão da sua especificidade para este tipo de função. No entanto, ainda que tenha simplificado a complexidade da criação do protótipo, não deixou menos trabalhoso, uma vez que, foi tentado deixar o protótipo com o maior detalhamento possível.

Diante disso, um dos maiores aprendizados durante a criação desta sprint, é a descoberta que existem ferramentas especializadas nas diferentes etapas do desenvolvimento de um projeto. Seja uma ferramenta para criação do ‘logo’ ou até para auxiliar na definição da paleta de cores. Por fim, foi possível observar como complemento do raciocínio que: nem sempre a melhor escolha é uma ferramenta única com funções generalistas.

2.2. Sprint 2

Nessa sprint, será apresentado os diagramas UML (*Unified Modeling Language*) criados para o desenvolvimento do aplicativo Mooras. Em termos práticos, a UML contempla uma série de notações para a construção de diagramas representando diferentes aspectos de um software, além de não estar presa a metodologias ou tecnologias específicas de desenvolvimento. Buscando assim um melhor entendimento daquilo que está sendo desenvolvido.

2.2.1. Solução

A seguir, os passos e resultados das etapas de elaboração dos diagramas escolhidos.

2.2.1.1. Evidências do Planejamento

Para registro do que foi planejado nessa Sprint, segue a captura de tela da ferramenta Trello, Figura 14, onde se encontra na lista a tarefa realizada nesta sprint.



Figura 14 – Tarefas realizadas no sprint 2 – Autoria Própria

2.2.1.2. Evidência da execução de cada requisito

Durante o processo de desenvolvimento foi realizado um esboço no papel, antes de usar softwares adequados para a criação dos diagramas. Esse esboço será documento e servirá de base para atividades de codificação das estruturas do sistema, bem como elaboração de testes das funcionalidades implementadas.

Na Figura 15, é possível observar o esboço dos diagramas: Entidade Relacionamento e do de Casos de Uso

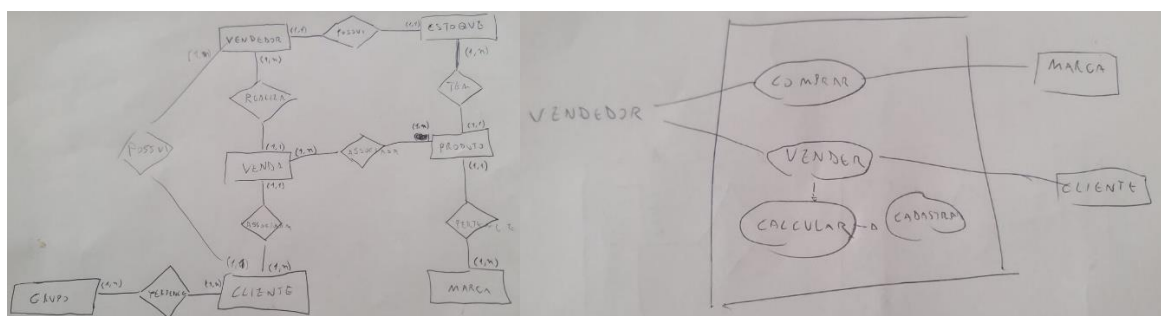


Figura 15 – Diagrama de Entidade Relacionamento e Diagrama de Casos de Uso – Autoria Própria

2.2.1.3. Evidência de solução

Neste subcapítulo é apresentado a solução para o ‘To Do’ da tarefa apresentada na lista do Trello. Vale ressaltar que todos os diagramas apresentados neste subcapítulo foram criados por meio da ferramenta ‘Word’.

Um diagrama entidade relacionamento (ER), Figura 16, é um tipo de fluxograma que ilustra como “entidades” (pessoas, objetos ou conceitos), se relacionam entre si dentro de um sistema. O diagrama facilita ainda a comunicação entre os integrantes da equipe, pois oferece uma linguagem comum utilizada tanto pelo analista, responsável por levantar os requisitos, e os desenvolvedores, responsáveis por implementar aquilo que foi modelado.

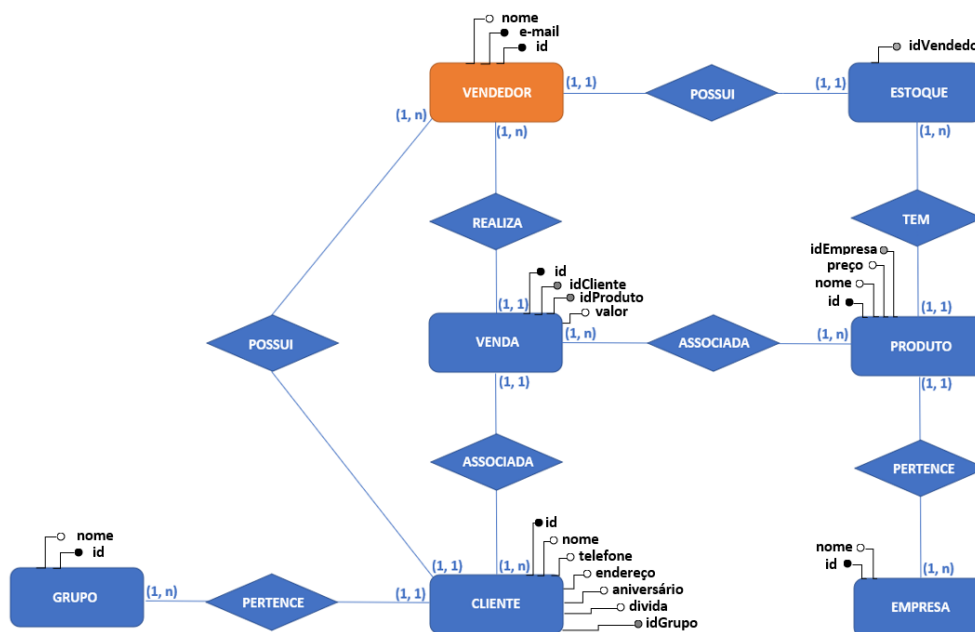


Figura 16 – Diagrama de Entidade Relacionamento – Autoria Própria

Já o objetivo do Diagrama de Casos de Uso, Figura 17, é demonstrar as diferentes maneiras que o usuário pode interagir com um sistema. O diagrama resume os detalhes dos usuários do seu sistema (também conhecidos como atores) e as interações deles com o sistema ajudando sua equipe a representar e discutir: os requisitos funcionais; cenários em que o sistema ou aplicativo interage com pessoas, organizações ou sistemas externos; metas que o sistema ou aplicativo ajuda essas entidades (conhecidas como atores) a atingir; e o escopo do sistema.

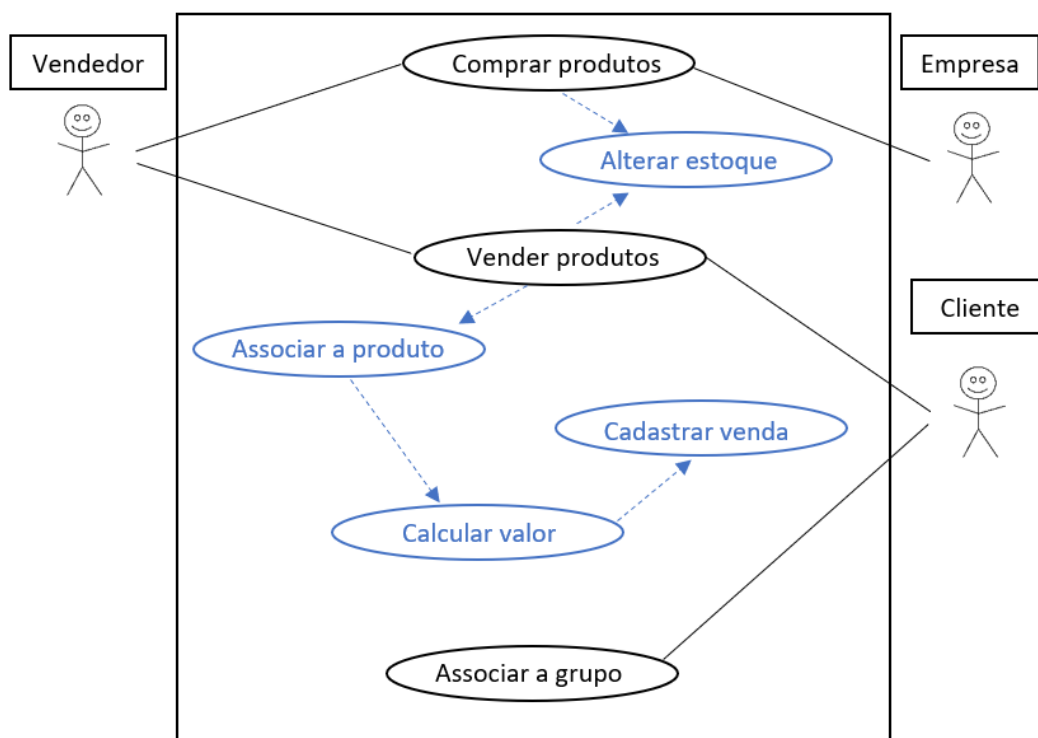


Figura 17 – Digrama de Casos de Uso – Autoria Própria

2.2.2. Lições Aprendidas

Esta etapa ajudou a minimizar os riscos no desenvolvimento do software. O uso de evidências providas a partir de estudos experimentais permite a caracterização de uma tecnologia antes de sua adoção em projetos de software na indústria de forma que seja possível determinar com níveis razoáveis de segurança sua viabilidade considerando cenários específicos de uso.

Destarte, a complexidade deste capítulo foi utilizar os conceitos e abordagens de engenharia da computação. Para isso foi utilizado ferramentas como o 'Word' para a criação dos diagramas.

Diante disso, um dos maiores aprendizados durante a criação desta sprint, é que documentar ideias para utilizar durante o desenvolvimento pode ser a melhor abordagem possível. Uma vez que, com o passar do tempo, e foco em algumas funcionalidades essenciais, outras funcionalidades pensadas no momento de ideação pode acabar por serem esquecidas no momento do desenvolvimento. A documentação ajuda a diminuir o escoamento dessas ideias.

2.3. Sprint 3

Nessa sprint, serão escolhidas as tecnologias para implementação e desenvolvimento do projeto, como: linguagem, frameworks, banco de dados, entre outros.

2.3.1. Solução

A seguir, os passos e resultados as etapas para definição das tecnologias escolhidas.

2.3.1.1. Evidencias do Planejamento

Para registro do que foi planejado nessa Sprint, segue a captura de tela da ferramenta Trello, Figura 15, onde se encontra na lista a tarefa realizada nesta sprint.

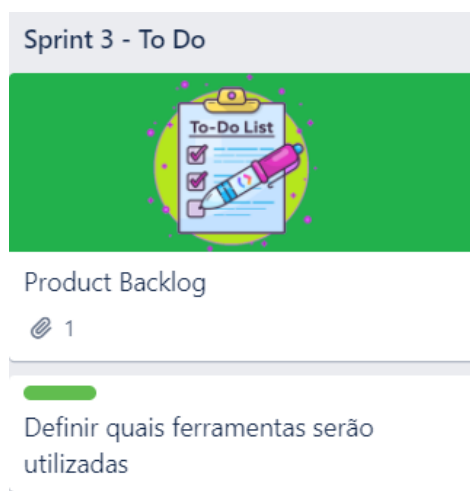


Figura 15 – Tarefas realizadas no sprint 3 – Autoria Própria

2.3.1.2. Evidência da execução de cada requisito

Foi definido que a melhor solução para essa problemática seria a criação de um aplicativo mobile. Ao se pensar em criar um aplicativo, alguns questionamentos devem ser respondidos. Entre eles é a escolha entre: um aplicativo nativo, um web App ou um aplicativo híbrido.

Alguns fatores devem ser analisados. Cada um é desenvolvido de uma maneira e deve ser pensado quais funções seu App oferecerá aos usuários, pois esse fator interfere diretamente no tipo de desenvolvimento. O aplicativo sempre deve oferecer uma boa experiência ao usuário, então deve se analisar aquele tipo que conseguirá suprir as necessidades deles.

- **Aplicativo Nativo:**

Por serem programados exclusivamente para o sistema operacional, o aplicativo nativo é mais confiável e rápido que os demais. Isso porque ele apresenta uma melhor experiência para o usuário ao conseguir utilizar todos os recursos oferecidos pelos smartphones.

Essa programação própria para o sistema operacional faz com que o desempenho do aplicativo nativo seja muito bom. Aplicativos nativos também possuem um maior tempo de utilização do que os demais por poderem funcionar sem conexão internet.

Porem o aplicativo nativo apenas funciona na plataforma que ele foi desenvolvido. Caso você o queira em mais de uma plataforma, pode optar por um plano de desenvolvimento que englobe tanto Objective-C como Javascript.

- **Web Apps:**

Web App não é um aplicativo real. Na verdade, é um site desenvolvido exclusivamente para dispositivos móveis. Possui uma programação que reconhece que o usuário está acessando por um smartphone e se adapta a ele.

Os códigos otimizados para dispositivos mobile oferecem uma boa experiência ao usuário. Quando a ideia é apenas apresentar conteúdo ou apenas ter presença mobile online, eles são uma excelente opção por serem mais baratos e fáceis de desenvolver. Toda sua programação é feita utilizando HTML5, CSS e Javascript.

Como é um site, o web App pode ser acessado de qualquer sistema operacional, desde que possua um navegador como o Google Chrome e o Safari instalado. Como você não faz o download do web App, ele não consome memória do seu celular.

O Web App necessita de conexão com a internet para ser acessado e não consegue utilizar todas as funcionalidades do seu dispositivo. São mais lentos que os aplicativos nativos por não serem integrados ao sistema operacional.

Ademais, os web apps não apresentam a mesma segurança que os outros tipos de aplicativos, podendo comprometer seu dispositivo.

- **Aplicativo Híbrido:**

O aplicativo híbrido, como o próprio nome já sugere, é uma mistura de um aplicativo nativo e um web App.

O aplicativo híbrido é construído na linguagem HTML5, CSS e Javascript, assim como o site mobile. Esse código é alocado dentro de um container, integrando as funcionalidades que o seu dispositivo oferece, oferecendo uma experiência melhor ao usuário que os web Apps. Assim, apenas uma parte do código nativo deverá ser escrito para esses apps. Isso possibilita que apenas uma parte do código seja reescrita caso queira oferecer o aplicativo para outra plataforma.

Eles estarão disponíveis para download nas lojas de aplicativos, oferecendo um canal de tráfego e download. Da mesma forma como o aplicativo nativo, o híbrido apresenta um custo de manutenção nas lojas.

Além disso, os híbridos são mais simples e, conseqüentemente, mais rápidos de desenvolver, e não se é necessário fazer o download sempre que uma atualização for feita em seu aplicativo. Mas sua utilização ainda dependerá de conexão com a internet e da velocidade da mesma, não funcionando tão rápido quanto um aplicativo nativo.

No entanto, isso não exclui a possibilidade de algumas funções do app funcionar offline, mas a internet é necessária para as atualizações do aplicativo.

De modo geral, é como se o aplicativo híbrido unisse o melhor de cada tipo de linguagem. Ele basicamente recebe uma linguagem com origem na web e é modelado com uma codificação nativa, onde ele pode ser usado tanto do seu smartphone como do seu notebook.

2.3.1.3. Evidência de solução

Dentro do contexto da pesquisa apresentado no subcapítulo anterior, a melhor solução é utilizar um aplicativo híbrido, uma vez que: aplicativos nativos não são multiplataformas; e aplicativos web precisam sempre de internet; já o híbrido funciona da forma que esta solução esta sendo idealizada, o uso principal da aplicação é o smartphone, mas, se preciso também pode ser utilizado por um notebook, além disso, o aplicativo tem a possibilidade funcionar offline e ficar online apenas para atualizar algumas funções.

Dentre as principais maneiras e tecnologias, as bibliotecas/frameworks mais comuns são: Cordova, PhoneGap, Xamarim, Ionic, Native Script, React Native e Flutter.

Dentre elas a escolhida para este projeto foi o React Native. Sendo um projeto open source mantido pelo Facebook e cumpre muito bem a promessa de desenvolver para Android e iOS com a mesma base de código.

Trata-se de uma nova tecnologia, mas já demonstra grande popularidade, sua aceitação no universo de desenvolvimento foi rápida. Ter sido lançada pelo Facebook é um dos motivos, mas permitir desenvolver utilizando apenas JavaScript na criação, manipulação, desenvolver elementos em tela, inputs de texto, botões e quaisquer itens do projeto, encapsular todo código nativo é a principal vantagem na escolha de tecnologia de projeto.

Quanto às tecnologias híbridas, o React Native cria componentes nativos em tela, não cria web views em HTML, JS, CSS, não sendo uma biblioteca híbrida, e assim, renderizando componentes nativos. Buscando desenvolver uma aplicação que dê a seus usuários uma experiência nativa com alta performance. Algumas de suas vantagens podem ser observadas na Figura 16.

Vantagens

- O React, traz um poderoso algoritmo de manipulação do DOM;
- O tamanho é extremamente leve;
- Usar React também deixa a aplicação mais robusta e fácil de implementar novas funcionalidades;
- Ferramentas de teste como o Jest, garantem que a aplicação funcione antes mesmo de ser feito um deploy para produção;
- O React não é um framework, você só instala o que é necessário para a aplicação.
- Não implementa DSL (Domain specific language)

Figura 16 – Vantagens da utilização do React – Autoria Própria

Sendo um dos principais motivos para a escolha desta tecnologia: a facilidade de uma forma imediata: a abordagem baseada em componentes, o ciclo de vida bem definido e o uso de JavaScript simplificado tornam o React muito simples de aprender. Em contrapartida na Figura 17 é possível observar algumas de suas desvantagens.

Desvantagens

- A renderização do template é toda feita em JavaScript então não temos loops, ifs, classes, inline no template;
- Bizarrices como, por exemplo, o react-if, ou o classnames;
- Se não tiver cuidado, com o React, você pode ter várias dependências pequenas espalhadas por toda a aplicação, a deixando muito pesada e lenta;
- O React em si é uma ferramenta simples e fácil de se programar, mas à medida que você vai adicionando funcionalidades e libs, sua complexidade aumenta muito.

Figura 17 – Desvantagens da utilização do React – Autoria Própria

Outra tecnologia de suma importância é o Firebase Realtime Database. Como a aplicação poderá funcionar de modo offline, ela deverá atualizar o banco de dados assim que for possível ficar online. No modo off-line:

- Aplicativos permanecem responsivos, mesmo sem conexão;
- O SDK do Firebase Realtime Database mantém os dados em disco;
- Quando a conectividade é restabelecida, o dispositivo cliente recebe as alterações não recebidas e faz a sincronização com o servidor.

2.3.2. Lições Aprendidas

O mundo tecnológico é extremamente grande e repleto de novas informações e oportunidades. Por isso, não basta manter-se atento às ferramentas que serão utilizadas, é importante também conhecer todas quem podem ser utilizadas, e qual atende melhor cada requisito do projeto.

Durante o processo de pesquisa para escolher a melhor tecnologia foi possível observar as diferenças entre elas e o melhor cenário para utilizá-las.

A etapa de desenvolvimento que vem a seguir será onde surgirá a oportunidade de por em pratica todo o aprendizado adquirido no decorrer da criação deste relatório.

A maior dificuldade deste capítulo foi reunir todas as informações adquiridas e compara-las para enfim decidir qual tecnologia melhor atendia o que era buscado. Ademais, a melhor maneira de continuar aprendendo é mantendo-se faminto pelo conhecimento e sempre buscando aprender mais.

2.4. Sprint 4

A sprint 4 será dedicada ao front-end do aplicativo, desenvolvido em react-native conforme especificado no subcapítulo 2.3.

2.4.1. Solução

A seguir, os passos e resultados das etapas para o desenvolvimento do front-end do Mooras.

2.4.1.1. Evidencias do Planejamento

Para registro do planejamento na Sprint, segue a captura de tela da ferramenta Trello, Figura 18, cuja a tarefa realizada se encontra na lista abaixo. É válido destacar acerca da tag de cor amarela, a qual foi apresentada nesta tonalidade pois a implementação do front-end não está concluída.

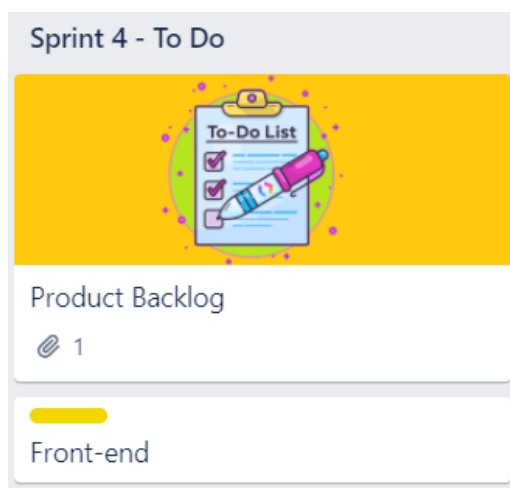


Figura 18 – Tarefas realizadas no sprint 4 – Autoria Própria

2.4.1.2. Evidência da execução de cada requisito

Para auxiliar na identificação das tarefas, realizou-se uma nova captura de tela do Trello. Na Figura 19, a cor verde em destaque tem como função representar os cards que já estão finalizados, os amarelos parcialmente completos e os vermelhos totalmente incompletos.

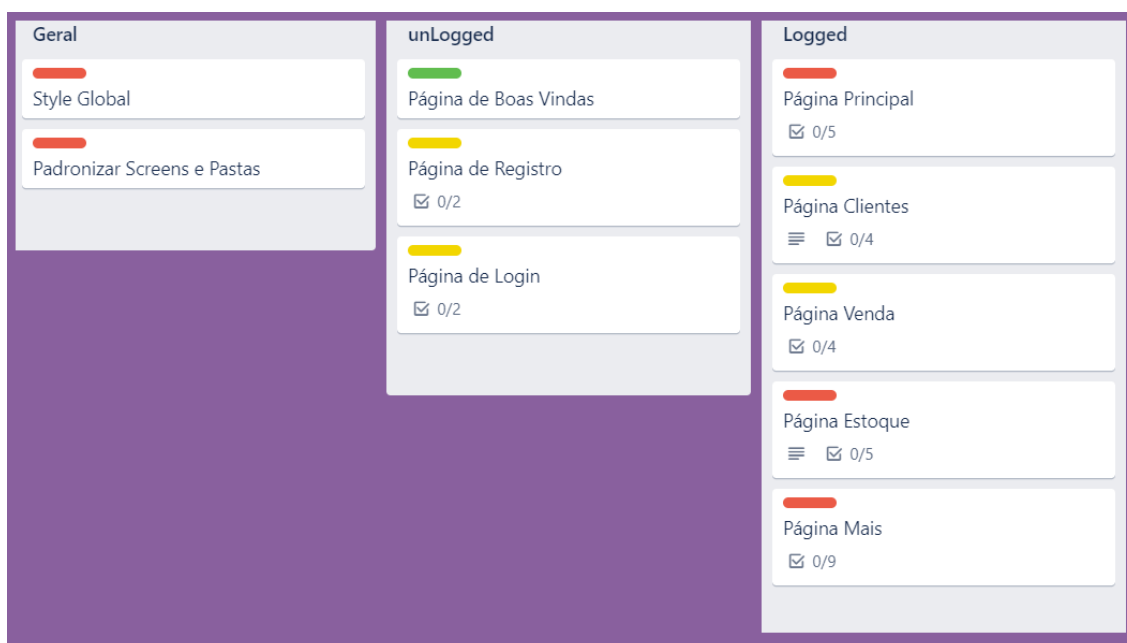


Figura 19 – Identificação das páginas completas – Autoria Própria

Dentro de cada card como exemplo da Figura 20, é possível observar os cartões ‘Página Clientes’ e ‘Página Estoque’. Na descrição consta os atributos dos mesmos, e em ‘O que falta?’, é referente a lista dos itens que necessitam de melhoramento ou implementação. Paralelamente, quando algo é cumprido da lista, ao invés de marcar no checkbox como ‘feito’, este item é retirado da lista tendo como objetivo proporcionar melhor organização. A exemplo disso, a ‘Listagem de clientes’ não mais se encontra na lista do que fazer da página de clientes.

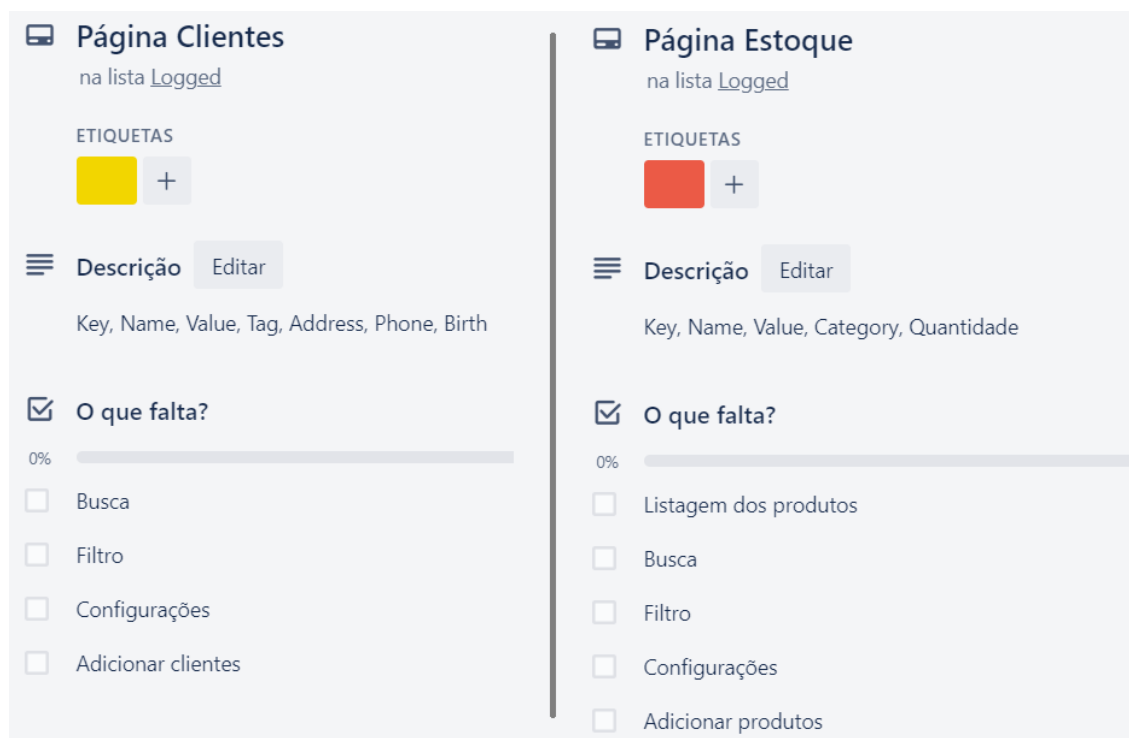


Figura 20 – Tarefas a serem realizadas – Autoria Própria

2.4.1.3. Evidência de solução

Inicialmente, é válido discorrer que este projeto teve uma grande incógnita no princípio, criar um projeto com ou sem expo. O Expo faz o papel de compilador do seu app: ele substitui a Android SDK e o Xcode por um método próprio, ou seja, não precisa de configuração prévia. Entretanto, o Expo 'limita' parte de alterar configurações ou inserir pacotes que modifiquem arquivos específicos do Android ou iOS. Como neste projeto não será preciso fazer alterações nas configurações de pacotes, o expo será de grande valia, no sentido de possibilitar suporte e promover acesso a inúmeras funções nativas.

```
<Stack.Screen options={{headerShown: false}} name="Home" component={HomeScreen} />
<Stack.Screen name="Register" component={RegisterScreen}
  options={{ title: 'Cadastrar',
    headerTintColor: '#fff',
    headerStyle: { backgroundColor: '#4c4c4c' }
  }}/>
<Stack.Screen name="Registered" component={RegisteredScreen}
  options={{ title: 'Entrar',
    headerTintColor: '#fff',
    headerStyle: { backgroundColor: '#4c4c4c' }
  }}/>
```

Figura 21 – Código das telas disponíveis para o usuário deslogado – Autoria Própria

No que se refere a codificação, o app.js é o arquivo de inicialização do projeto, nele foi colocado apenas as opções de navegação entre telas. Utilizando a biblioteca 'React Navigation' foram criadas duas estruturas, sendo a primeira Figura 21, para quando o usuário estiver deslogado e a segunda Figura 22 para quando o usuário estiver logado.

```
<Tab.Screen options={{ title: 'Principal' }} name="Main" component={MainScreen} />
<Tab.Screen options={{ title: 'Clientes' }} name="Clients" component={ClientsScreen} />
<Tab.Screen name="AddSale" component={AddSaleScreen}
options={() => ({
  tabBarIcon: () => (
    <View style={styles.iconTabRound}>
      <Ionicons name="ios-add" size={40} color="#fff" />
    </View>
  ),
})} />
<Tab.Screen options={{ title: 'Estoque' }} name="Stock" component={StockScreen} />
<Tab.Screen options={{ title: 'Mais' }} name="More" component={MoreScreen} />
```

Figura 22 – Código das telas disponíveis para o usuário logado – Autoria Própria

Na Figura 23 é possível observar as telas disponíveis para quando o usuário ainda não fez o login, a primeira tela é a de 'Boas-vindas', por meio dela pode-se acessar a tela 'Cadastrar' e a tela 'Entrar'. Na tela 'Cadastrar' será onde o usuário criará suas credenciais para acessar o aplicativo na tela 'Entrar'.

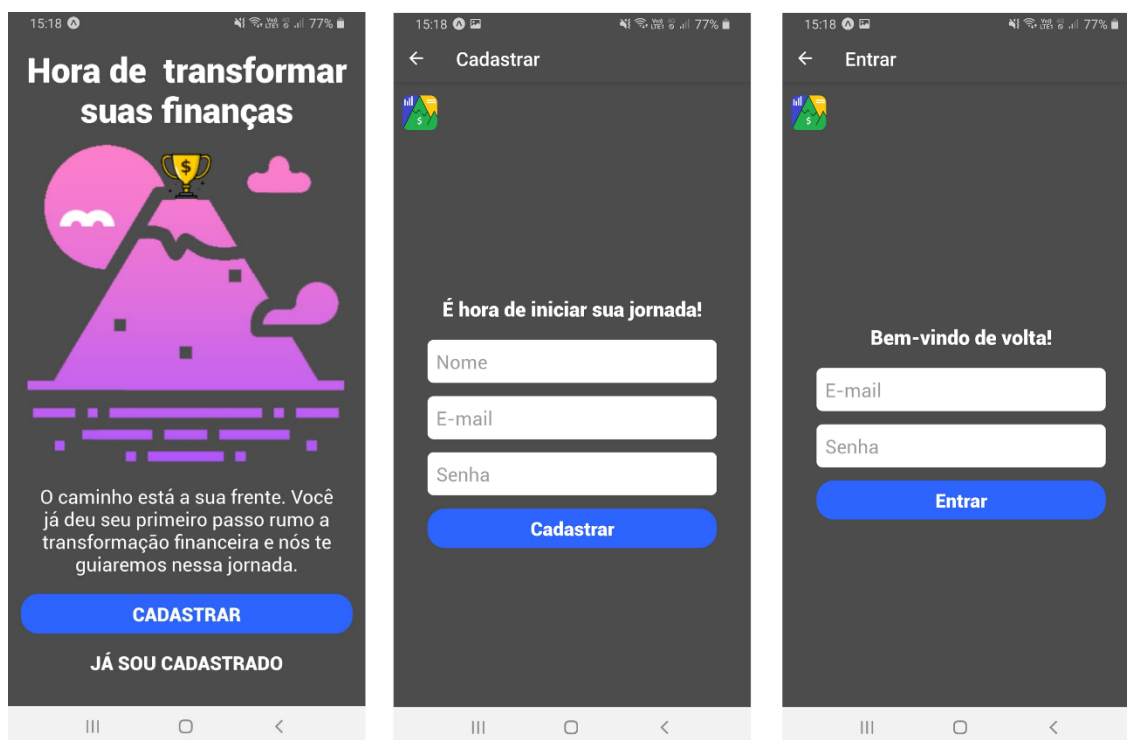


Figura 23 – Telas disponíveis para o usuário logado – Autoria Própria

Tanto a tela de cadastro quanto a de login, possuem características especiais. Como é possível observar na Figura 24, ao abrir o teclado os inputs não perdem o foco, são jogados para cima da tela para que possam ser visualizados o tempo todo devido as configurações nativas da 'KeyboardAvoidingView', além disso, possuem animações de aparecimento. Os campos tem a

opacidade alterada gradativamente, de transparente para solido, e o ícone do aplicativo sobe do início da tela até a parte mais alta, essas funcionalidades se devem a biblioteca ‘react-native-animated’.

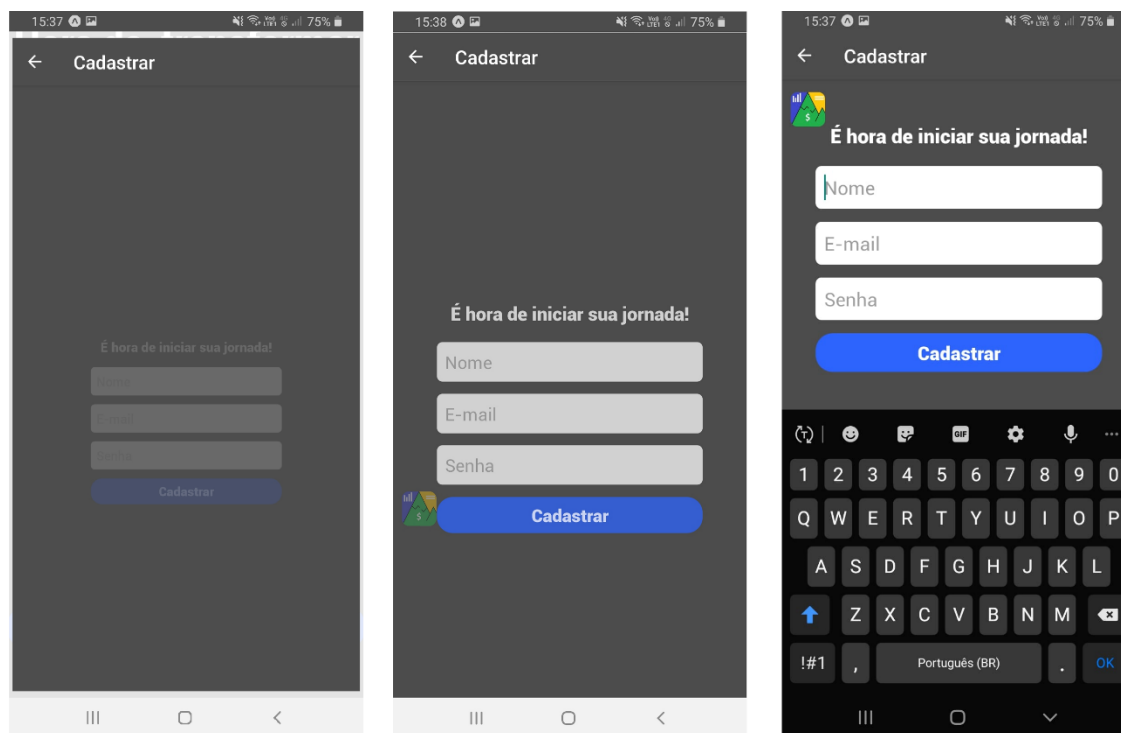


Figura 24 – Telas disponíveis para o usuário deslogado – Autoria Própria

Já as telas disponíveis para quando o usuário estiver logado podem ser observadas na Figura 25. Nesta sprint, apenas as telas ‘Clientes’ (acessado pela segunda aba do Tab.Navigator) e a tela ‘Nova venda’ (acessada pelo botão central - redondo e azul) foram desenvolvidas.

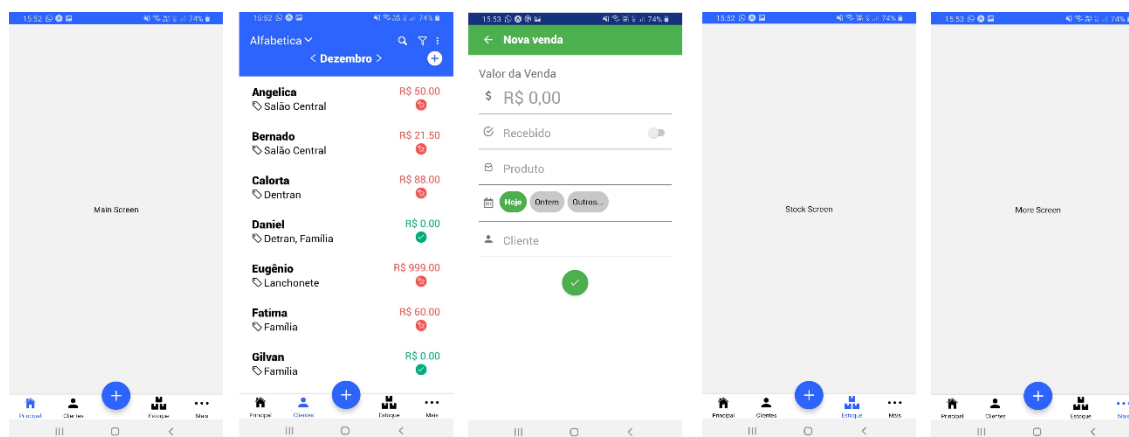


Figura 25 – Telas disponíveis para o usuário logado – Autoria Própria

A lista da tela ‘Clientes’ foi implementada utilizando ‘FlatList’ e sendo preenchida conforme a Figura 26. Nesta sprint, ainda não foi implementado uma forma de adicionar os clientes, tendo em vista que a tela para o mesmo ainda não foi desenvolvida. Todavia, o botão que fará tal ação fica no canto inferior direito do header, ao lado do nome ‘Dezembro’.

```
const [client, setClient] = React.useState([
  {key: 1, name: 'Angelica', value: 50.00, tag: 'Salão Central'},
  {key: 2, name: 'Bernado', value: 21.50, tag: 'Salão Central'},
  {key: 3, name: 'Calorta', value: 88.00, tag: 'Dentran'},
  {key: 4, name: 'Daniel', value: 0.00, tag: 'Detran, Família'},
  {key: 5, name: 'Eugênio', value: 999.00, tag: 'Lanchonete'},
  {key: 6, name: 'Fatima', value: 60.00, tag: 'Família'},
  {key: 7, name: 'Gilvan', value: 0.00, tag: 'Família'},
]);
```

Figura 26 – Preenchimento de dados da tela Cliente – Autoria Própria

A página 'Nova venda' foi criada dentro de um Modal (uma maneira básica de apresentar conteúdo acima de uma tela – como se fosse uma tela sobre outra). As futuras páginas 'novo cliente', 'novo produto' e todas as funcionalidades da página 'Mais'

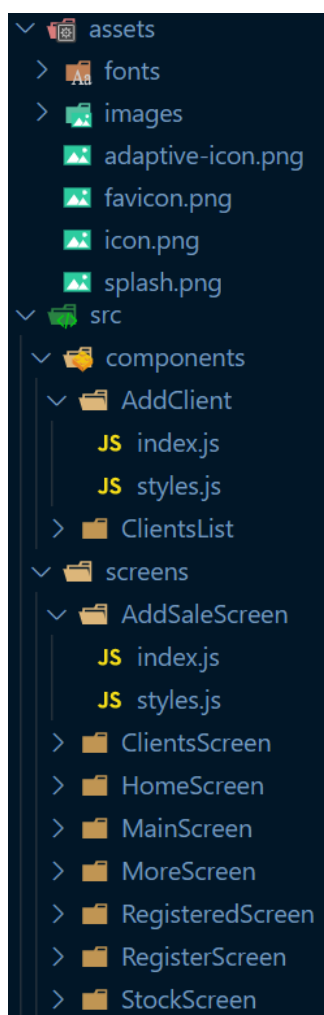


Figura 27 – Demonstração da estrutura dos arquivos – Autoria Própria

também serão desenvolvidos nessa estrutura. Nesse segmento, uma nova venda tem que está associada a um cliente, a um produto, ter um preço, saber se já foi pago e a data da compra, todas essas características foram sanadas na página em questão.

Em relação ao estruturamento de pastas e organização dos arquivos, a Figura 27 retrata a forma que está sendo feita. Todas as páginas principais dentro da pasta 'screens' no src e as páginas secundarias dentro da página 'components' no src, cada pasta de página possui o seu respectivo 'index.js' contendo o código bruto em si, e o style.js para permitir a separação da estilização. Desse modo, a organização dos arquivos fica melhor e não é necessário mudar as importações

O que é comum de toda aplicação, tanto web quando mobile, é geralmente seguir um padrão de cores, espaçamentos, fontes, etc. Nesses casos o mais correto é separar esses estilos genéricos em uma pasta alternativa "global" ou "pasta raiz". Essa pasta fica responsável por armazenar qualquer estilo que irá ser utilizado em mais lugares do projeto, evitando redundâncias. Contudo, no momento ainda não foi desenvolvido um style.js global, porém está dentro do escopo de desenvolvimento.

Em relação aos arquivos externos, visto também na Figura 27, todos foram colocados em 'assets', desde a fonte das letras (onde foi usado a biblioteca 'expo-font') até as imagens, como a montanha na tela de 'Boas-vindas'

2.3.2. Lições Aprendidas

Mesmo com o planejamento anterior, ainda assim foi necessário adicionar outras páginas que não foram retratadas durante o protótipo de alta fidelidade, tais como: a página para o usuário se registrar, a página para o usuário fazer login e a página de adicionar uma nova venda. Todas essas páginas já vinham sendo idealizadas, mas não tinham sido formalmente retratadas neste relatório.

É plausível afirmar, que ao longo da experiência adquirida durante a graduação, este projeto é tido como “primeiro” contato com desenvolvimento web e mobile. Nesse ínterim, a inexperiência dificultou o andamento do projeto, pois era previsto finalizar a aplicação do front-end nesta etapa, não obstante, foi possível apenas concluir o front-end da parte de registro e login, bem como a listagem de cliente e cadastro de uma nova venda.

Nesta sprint, foi possível aprender diversos conceitos e sentir a “pegada” do react-native, a partir disso, é concebível estimar com mais precisão o tempo de implementação de cada modulo. Na próxima Sprint é pretendido ter finalizado todo o front-end e back-end de cliente e estoque, admitindo ter um MVP, o aplicativo poderá ser testado e avançar para outras etapas.

2.5. Sprint 5

A sprint 5 será principalmente dedicada ao back-end do aplicativo, onde foi desenvolvido um API REST NodeJS com amparo do Express e Mongo.

2.5.1. Solução

A seguir, os passos e resultados das etapas para o desenvolvimento do back-end do Mooras.

2.5.1.1. Evidencias do Planejamento

Para registro do planejamento na Sprint, segue a captura de tela da ferramenta Trello, Figura 28, cujas tarefas realizadas se encontram na lista abaixo. É válido destacar que as tag's de cor verde significam que a tarefa foi finalizada, e cerca das tag's de cor amarela, a qual foi apresentada nesta tonalidade pois a implementação foi iniciada, mas não concluída.

É importante destacar que a tarefa ‘Desenvolver estrutura inicial do projeto (pasta e design pattern)’ foi iniciada na Sprint anterior, contudo, ainda haviam melhorias a serem realizadas, por essa razão a tarefa foi atribuída a esta Sprint e finalizada. Em relação as tarefas ‘Front-end’ e ‘Desenvolver de acordo com as funcionalidades e layout definidos’, foi implementado a página de cadastro de cliente e as validações e autenticações necessárias para o login. As três ultimas tarefas da Figura 28 tem relação com o back-end, que foi o foco dessa sprint e que terão seu desenvolvimento descrito nos subcapítulos subsequentes.

Ademais, a Figura 28 descreve as tarefas do Product Backlog, apesar de nesta Sprint também ter avanço no Concept Backlog, foi preferível minuciar as tarefas do primeiro, pois descreveria melhor o avanço no desenvolvimento.

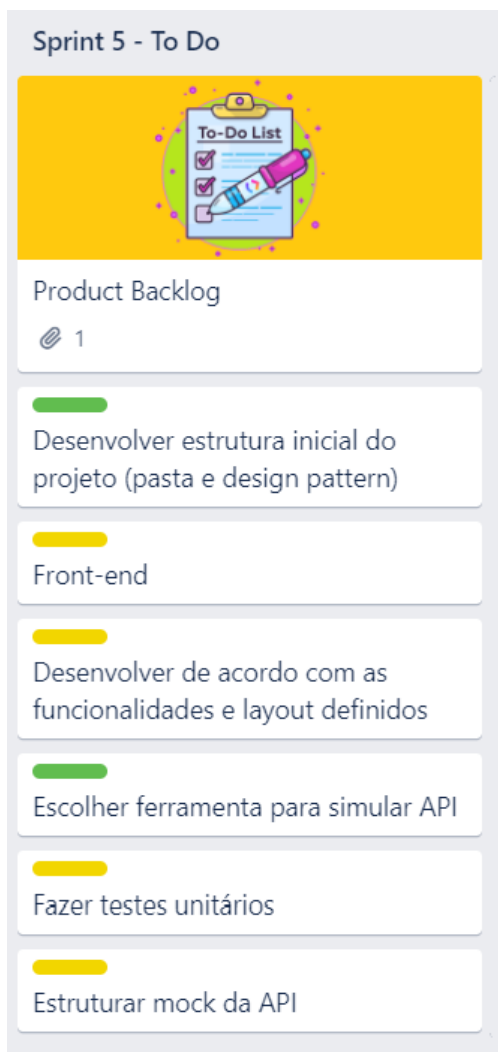


Figura 28 – Tarefas realizadas no sprint 5 – Autoria Própria

2.5.1.2. Evidência da execução de cada requisito

Tarefas restantes		
Sprint 6	Integrar aplicação front-end com a API Registro Autenticação Cliente	dia 23/01 a 25/01
	Página do cliente Config. Botões header Add client : quebra de linha nas tags // botao voltar do cel fechar modal Exibir todos os dados do cliente Conectar com vendas	dia 30/01 a 01/02
	Página do estoque Copiar config. Cliente Configurar API	dia 06/02 a 08/02
	Página adicionar venda Configurar API Refatorar inputs da página Ao finalizar voltar para clientes	dia 13/02 a 15/02
	Página de mais opções	
Sprint 7	Página principal Mobile / Web	dia 20/02 a 01/03

Figura 29 – Cronograma de desenvolvimento das tarefas restantes após esta Sprint – Autoria Própria

Para auxiliar no desenvolvimento das tarefas uma tabela foi montada com o cronograma de desenvolvimento de cada parte da aplicação, Figura 29. É possível perceber que a Figura não é possível ver o que foi desenvolvido nesta Sprint, já que as partes finalizadas são removidas da tabela para possibilitar que as restantes possam ser melhor detalhadas.

2.5.1.3. Evidência de solução

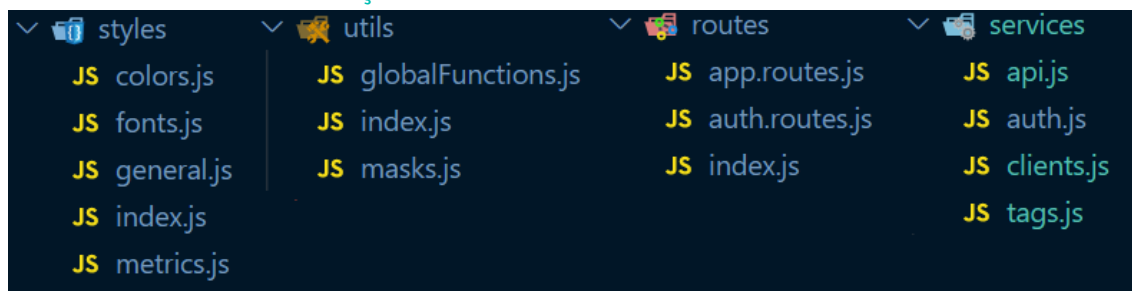


Figura 30 – Estruturação de pastas – Autoria Própria

Primeiramente, é importante ressaltar as alterações feitas no projeto entre esta Sprint e a anterior. Uma dessas foi na estruturação de pastas, conforme pode ser observado na Figura 30. A pasta 'style' armazena todos os estilos que são comuns a vários trechos de código, como exemplo: vários botões que possuem a mesma tonalidade de cor, '#00AF80' (uma tonalidade de verde em hexadecimal), ao invés de cada botão possuir como característica o valor dessa cor, é possível criar uma variável (exemplo 'green_check'), e atribuir o valor da cor a esta variável, e colocar esta variável como característica do botão, dessa forma, se em algum momento for preciso alterar a tonalidade desta cor, não é necessário alterar de botão em botão, basta alterar o valor da variável 'green_check', conforme pode ser observado na Figura 31 o conteúdo da subpasta 'colors'. A subpasta 'fonts' funciona da mesma forma, porém para os estilos de letras (arial, calibre, roboto e outros), a subpasta 'metrics' é utilizada para as medidas (margem, tamanho de fonte, etc.), pôr fim a subpasta 'general' é utilizada para componentes em comum (botões, caixas de entradas, entre outros) que possuem a mesma estilização.

```
const colors = {
  white_pattern: '#ffffff',
  black_pattern: '#000000',
  blue_pattern: '#2d64fd',
  grey_pattern: '#969696',
  green_pattern: '#60a962',
  ...
  red_check: '#FB5252',
  green_check: '#00AF80',
  grey_check: '#C2C2C2',
  ...
  backgroundColorHomeScreen: '#4c4c4c'
};
```

Figura 31 – Conteúdo da pasta 'colors' – Autoria Própria

A pasta 'utils' da Figura 30, tem a função de armazenar funções importantes e que serão utilizadas em todo o projeto, a exemplo das 'globalFunctions' que armazenam entre outras, as funções de seleção de tag e input. Assim como a subpasta 'masks' que armazena as funções que mascaram as caixas de entrada dependendo do que se espera receber.

Acresce a isso existem as pastas 'routes' e 'services', na primeira pasta é onde o direcionamento do usuário é realizado, dependendo se esta logado ou não. Já a pasta services é onde serão feitas a conexão entre o fron-end e a API.

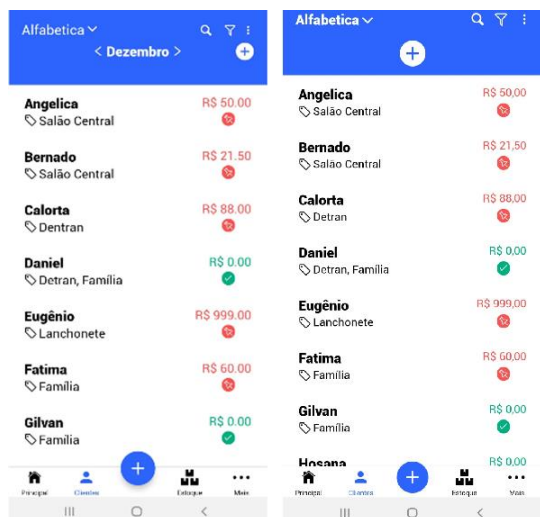


Figura 31 – Alteração da página cliente – Autoria Própria

Outra alteração foi no header da página ‘Clientes’ que pode ser observado na Figura 32, retirou-se o controle mensal desta página pois a necessidade desse tipo de controle é em vendas, os clientes não mudarão de mês para mês, com isso o botão de adicionar um novo cliente foi centralizado.

Já sobre novas implementações, foi desenvolvida o modal para adicionar novo cliente Figura 32. Em comparação com ReactJS o React Native possui um número mais limitado de eventos, por isso foi necessário criar funções customizadas com mascaras para cada input. Também na imagem 32, é possível ver o antes e o depois da utilização das máscaras no modal.

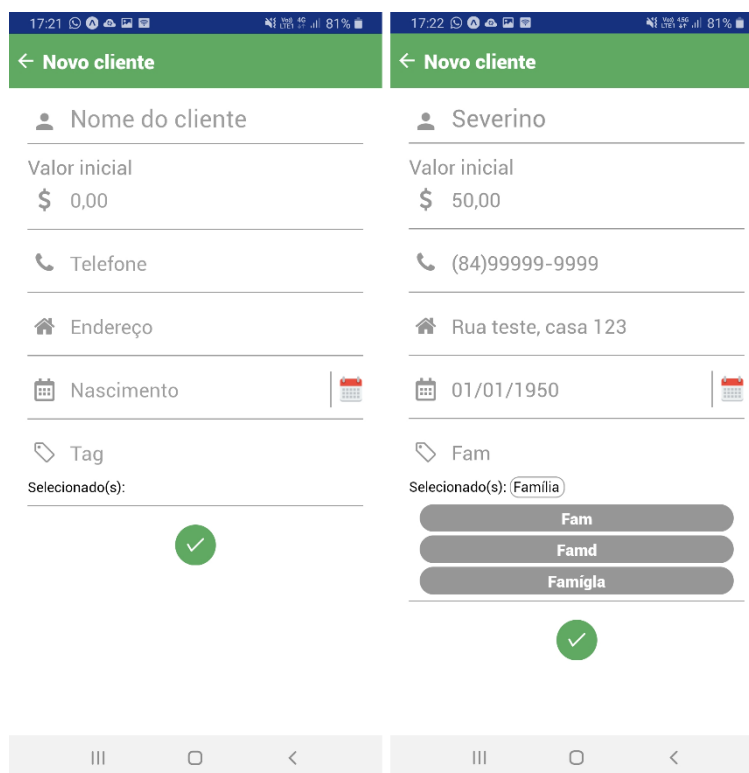


Figura 32 – Modal para adicionar cliente – Autoria Própria

A exemplo o input de telefone, quando o usuário digita 8499999999999, o valor do campo é atualizado em tempo real para (84)99999-9999. Estas mascaras, Figura 33, foram desenvolvidas utilizando regex, ou seja, são utilizados padrões para determinar ocorrências em cadeias de caracteres de interesse. No input da tag, ao digitar alguma palavra é realizado uma filtragem das tags já existentes, a exemplo da Figura 32, onde o usuário digita ‘fam’ e todas as tag’s já cadastradas aparecem, ao clicar no botão cinza ela é selecionada (a exemplo da tag família).

```
function maskPhone(value) {
  value = value.replace(/\D/g, "");
  value = value.replace(/^(\d{2})(\d)/g, "($1)$2");
  value = value.replace(/(\d)(\d{4})$/g, "$1-$2");
  return value;
}

function maskCurrency(value) {
  value = value.replace(/\D/g, "");
  value = value.replace(/(\d)(\d{2})$/g, "$1,$2");
  value = value.replace(/(=(\d{3})+(\d))\B/g, ".");
  return value;
}

function maskBirth(value) {
  value = value.replace(/\D/g, "");
  value = value.replace(/^(\d{2})(\d)/g, "$1t$2");
  value = value.replace(/^(\w{5})(\d)/g, "$1t$2");
  value = value.replace("t", "/");
  value = value.replace("t", "/");
  return value;
}
```

Figura 33 – Mascaras de telefone, moeda e data de nascimento – Autoria Própria

Ainda se tratando do fron-end, foi desenvolvido toda a parte de autenticação do usuário, o contexto 'authProvider', Figura 34, possui as funções de validação.

```
export const AuthProvider = ({ children }) => {
  const [user, setUser] = React.useState(null);
  const [loading, setLoading] = React.useState(true);

  React.useEffect(() => {
    async function loadStorageData() {
      const storageUser = await AsyncStorage.getItem('@RNAAuth:user');
      const storageToken = await AsyncStorage.getItem('@RNAAuth:token');

      if (storageUser && storageToken) {
        setUser(JSON.parse(storageUser));
        setLoading(false);
      } else if (!storageUser && !storageToken) {
        setLoading(false);
      }
    }

    loadStorageData();
  }, []);

  async function signIn() {
    const response = await auth.signIn();
    setUser(response.user);

    await AsyncStorage.setItem('@RNAAuth:user', JSON.stringify(response.user));
    await AsyncStorage.setItem('@RNAAuth:token', response.token);
  }

  function signOut() {
    AsyncStorage.clear().then(() => {
      setUser(null);
    });
  }

  return (
    <AuthContext.Provider value={{ signed: !!user, user, signIn, signOut, loading }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Figura 34 – AuthProvider – Autoria Própria

A exemplo das funções a direita da Figura 34, a de signIn para o usuário entrar no aplicativo, e a função de sigOut para deslogar. O useEffect a esquerda, é utilizado para quando o usuário entrar no aplicativo verificar se esta logado e ser redirecionado para as rotas correspondentes, Figura 35.

```
return signed ? <AppRoutes /> : <AuthRoutes />;
```

Figura 35 – Redirecionamento de rotas – Autoria Própria

Como a conexão entre o front-end e a API ainda não foi estabelecida, para efeitos de desenvolvimento foi utilizado um mock de estrutura, citada na Figura 36 para testar as autenticações.

```
export function signIn() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve({
        token: 'dh2ur328uf8hgiejgoew9i23',
        user: {
          name: 'Daniel',
          email: 'daniel@gmail.com',
        },
      });
    }, 2000);
  });
}
```

Figura 36 – Redirecionamento de rotas – Autoria Própria

Já se tratando do back-end, foi desenvolvido um API REST utilizando NodeJS e Express. O Express é um framework para o Node e torna o desenvolvimento das aplicações mais rápidas e fácil, além de facilitar o roteamento da aplicação. As solicitações desenvolvidas podem ser observadas na Figura 37. Para o usuário foi desenvolvido a autenticação, o registro e as funcionalidades de mudança de senha (reset e forgot). Para o cliente e tag foi desenvolvido a de listagem de todos, listagem por id, criação, atualização e deletar. É possível observar na Figura 38 a implementação para registro do usuário

Auth	
POST	Authenticate
POST	Register
POST	Forgot Password
POST	Reset Password
Client	
GET	List
GET	Show
POST	Create
PUT	Update
DEL	Delete
Tag	
GET	List
GET	Show
POST	Create
PUT	Update
DEL	Delete

Figura 37 –
Solicitações –
Autoria Própria

```
router.post('/register', async (req, res) => {
  const { email } = req.body;

  try {
    if (await User.findOne({ email })) {
      return res.status(400).send({ error: 'User already exists' });
    }

    const user = await User.create(req.body);
    user.password = undefined;

    return res.send({
      user,
      token: generateToken({ id: user.id }),
    });
  } catch (err) {
    return res.status(400).send({ error: 'Registration failed' });
  }
});
```

Figura 38 – Registro do usuário – Autoria Própria

```
module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader)
    return res.status(401).send({ error: 'No token provided' });

  const parts = authHeader.split(' ');

  if (!parts.length === 2)
    return res.status(401).send({ error: 'Token error' });

  const [scheme, token] = parts;

  if (!/^Bearer$/i.test(scheme))
    return res.status(401).send({ error: 'Token malformed' });

  jwt.verify(token, authConfig.secret, (err, decoded) => {
    if (err) return res.status(401).send({ error: 'Token invalid' });

    req.userId = decoded.id;
    return next();
  });
};
```

Figura 39 – Verificação do token – Autoria Própria

Para realizar a autenticação é necessário fazer a verificação do token, se essa verificação for feita direto é necessário processamento da máquina. Por essa razão as verificações da Figura 39 são necessárias, por serem simples não consomem quase nada de processamento e é saudável para o backend, antes de fazer uma verificação mais pesada é importante validar o máximo possível.

```
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    require: true
  },
  email: {
    type: String,
    require: true,
    unique: true,
    lowercase: true
  },
  password: {
    type: String,
    require: true,
    select: false
  },
  passwordResetToken: {
    type: String,
    select: false
  },
  passwordResetExpires: {
    type: Date,
    select: false
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});
```

Figura 40 – Esquema do usuário – Autoria Própria

Para a criação do usuário, assim como do cliente e da tag, foi descrito o esquema que auxilia na concepção dos mesmos. A Figura 40 demonstra o esquema de usuário, evidenciando o tipo (type) de cada variável, as variáveis obrigatórias possuem o 'require' como verdadeiro e a que possui unique funciona como uma variável identificadora. É importante notar que o password (senha) tem o 'select' marcado como falso, pois não é para ser transmitido nas chamadas.

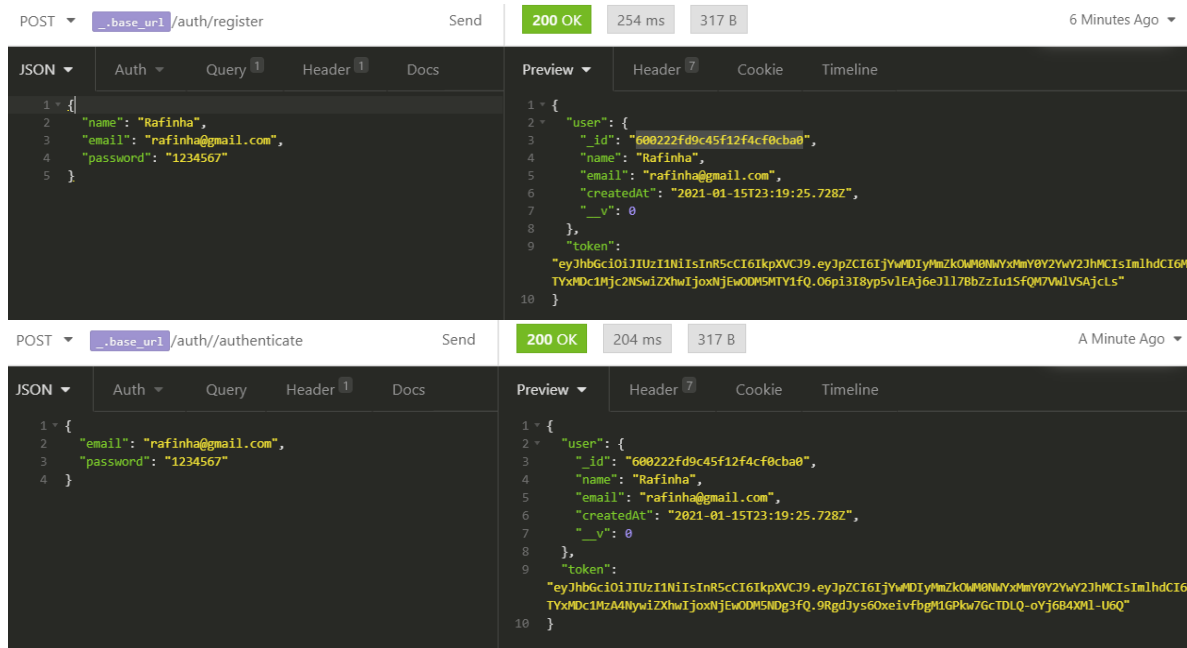


Figura 40 – Chamada de registro e autenticação do usuário – Autoria Própria

Na Figura 40 é demonstrado as chamadas de registro e autenticação do usuário, esse token é necessário porque cada usuário possui seus próprios clientes e devem ter acesso somente aos dados dos mesmos.

Na Figura 41 é realizado uma listagem por id dos clientes, é importante notar que um cliente pode possuir nenhuma, uma ou mais tag's. Para fazer o registro de um cliente o usuário precisa passar um token valido, para isso é necessário um middleware para autenticar o token, este pode ser visto na Figura 39. No cliente é salvo o id do usuário, para que um fique associado ao outro. Apesar da Figura ao lado demonstrar o objeto completo do usuário (id, name e outros), no cliente apenas é salvo o id, e por meio do método 'populate' do mongoose que o 'user' é preenchido na chamada.

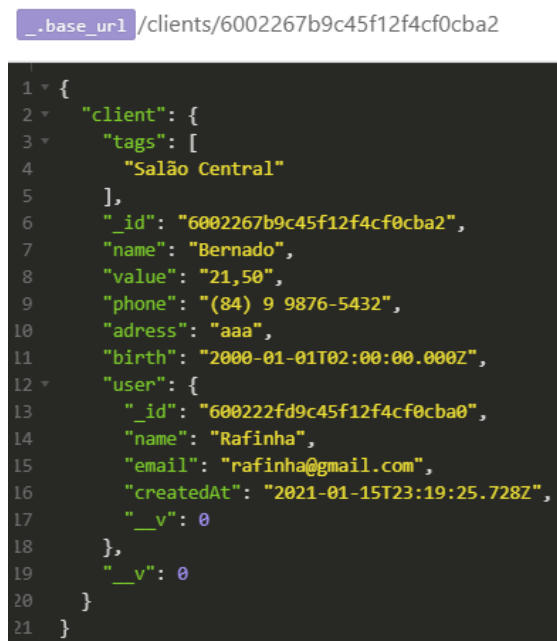


Figura 41 – Listagem de cliente por id – Autoria Própria

As ferramentas utilizadas para a realização dos testes foi o Insomnia para a iteração com APIs e o Mongo para acesso aos dados de forma mais amigável.

2.5.2. Lições Aprendidas

Na Sprint anterior foi relatado que a parte do front-end e de back-end de cliente e estoque seriam finalizados nesta sprint, não foi possível cumprir o prazo. Um dos motivos foi que durante o desenvolvimento alguns arquivos foram modificados e outros excluídos permanentemente, e tiveram que ser refeitos, ocasionando um grande período de retrabalho. Para que este problema não se repita, foi realizado o upload do projeto no github, que pode ser encontrado no endereço eletrônico: “<https://github.com/Daniel-FC/Mooras.git>”.

Outro motivo foi a tentativa frustrada de utilizar o Firebase (mencionado seu uso na Sprint 3), que se mostrou uma implementação complexa para o uso de suas funcionalidades. Sua utilização poderia ter ajudado na autenticação do usuário e no armazenamento off-line e on-line de dados.

Conforme foi explicado no subcapítulo ‘2.5.1.2’, na próxima Sprint é pretendido finalizar todo o front-end e back-end de cliente, estoque e vendas. Admitindo ter um MVP, o aplicativo poderá ser testado e avançar para outras etapas.

2.6. Sprint 6

A sprint 6 será a reformulação da API devido a complicações na conexão do front-end com o back-end e a continuação da implementação das telas essenciais para uso do aplicativo.

2.6.1. Solução

A seguir, os passos e resultados das etapas para o desenvolvimento do back-end e front-end do Mooras.

2.6.1.1. Evidências do Planejamento

Para registro do planejamento na Sprint, segue a captura de tela da ferramenta Trello, Figura 42, cujas tarefas realizadas se encontram na lista ao lado. É válido destacar que as tag’s de cor verde significam que a tarefa foi finalizada, e cerca das tag’s de cor amarela, a qual foi apresentada nesta tonalidade pois as funcionalidades ainda se encontram em desenvolvimento.

Vale destacar que nesta Sprint a implementação da API foi finalizada, assim como a integração da mesma com o front-end.

Devido a complicações na integração e sugestão do orientador da disciplina houve uma refatoração do código com o objetivo de desacoplar o projeto do MongoDB.

Após esse processo foi possível realizar todos os testes unitários, tanto utilizando a ferramenta Insomnia quanto direto do aplicativo Mooras, mais para frente será melhor detalhado todos esses processos.



Figura 42 – Tarefas realizadas no sprint 6 – Autoria Própria

2.6.1.2. Evidência da execução de cada requisito

Para auxiliar na identificação das tarefas do front-end, realizou-se uma nova captura de tela do Trello. Na Figura 43, a cor verde em destaque tem como função representar os cards que já estão finalizados, os amarelos parcialmente completos e os vermelhos totalmente incompletos.

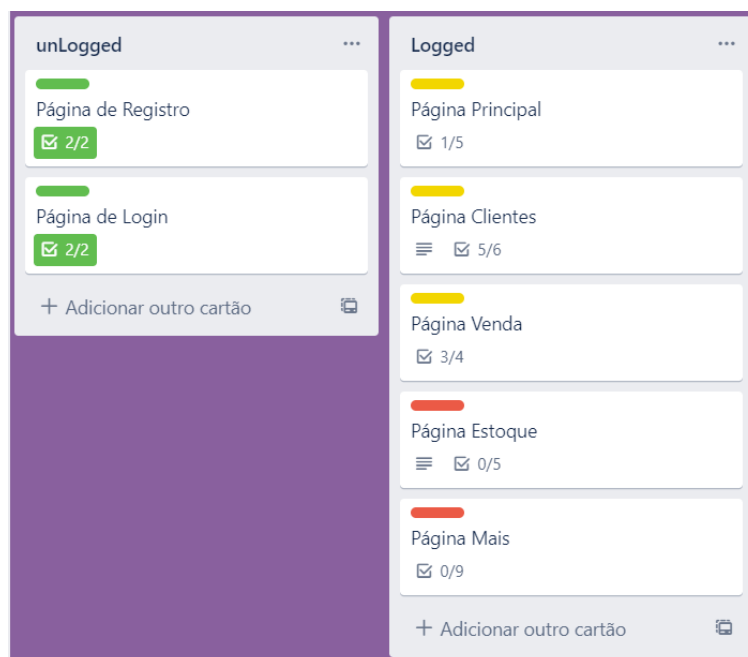


Figura 43 – Identificação das páginas completas – Autoria Própria

Comparando a Figura 43, com a Figura 19, é possível observar o grande avanço no projeto neste período de tempo. Foram finalizadas as páginas 'Registro' e 'Login', e as páginas 'Cliente', 'Venda' e 'Principal' tiveram mudanças significativas.

2.6.1.3. Evidência de solução

Primeiramente é preciso explicar o porque foi necessario refatorar a API: na existencia de uma dificuldade em integrar a API com a aplicação, foi sugerido pelo professor orientador a utilização do 'MongoDB Atlas' para desacoplação das dependencias do MongoDB do projeto. Entretanto foi utilizado uma outra estratégia.

```
const express = require('express');
const fs = require('fs');
const util = require('util');
const authMiddlewares = require('../middlewares/auth');

const router = express.Router();
const promisify = util.promisify;
const writeFile = promisify(fs.writeFile);
const readFile = promisify(fs.readFile);

const tagsDatabase = 'src/database/tags.json';

router.use(authMiddlewares);
```

Observando a imagem 44 é possível notar que: por meio do 'promisify' (seta amarela) da biblioteca 'util' (seta verde) é possível ler e escrever em um arquivo. No caso da imagem, o arquivo em questão é o 'tags.json'.

Nesse novo metodo de implementação, os dados são salvos em arquivos dentro da propria API, conforme Figura 45.

Figura 44 – Biblioteca de edição de arquivo – Autoria Própria

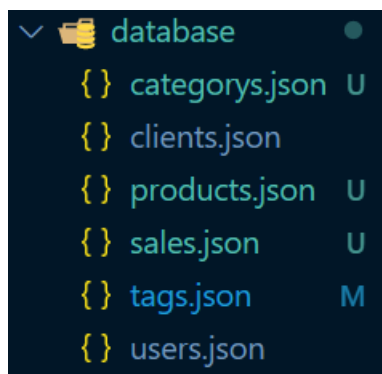


Figura 45 – Organização de pastas do método de salvar informações – Autoria Própria

A exemplo da Figura 46, do método 'post' de Products, é possível observar as etapas do processo. Perceba que além das informações contidas no body da requisição, são salvas o 'id' do usuário, o 'id' do produto, e a informação do próximo 'id' do produto a ser gerado.

```
router.post('/', async (req, res) => {
  try {
    const data = JSON.parse(await readFile(productsDatabase, 'utf8'));
    let product = await req.body;

    product = { id: data.nextId++, ...product, user: req.userId };
    data.products.push(product);
    await writeFile(productsDatabase, JSON.stringify(data));

    return res.send({ product });
  } catch (err) {
    return res.status(400).send({ error: 'Error creating new product' });
  }
});
```

```
1 {
2   "nextId":3,
3   "products":[
4     {
5       "id":1,
6       "name":"Egeo On Yoy",
7       "value":"114,90",
8       "categorys":["O Boticário"],
9       "quantity":"1",
10      "user":2
11    },
12    {
13      "id":2,
14      "name":"Glamour",
15      "value":"107,90",
16      "categorys":["O Boticário"],
17      "quantity":"4",
18      "user":2
19    }
20  ]
21 }
```

Figura 46 – Método 'post' da Tag – Autoria Própria

Já a Figura 47 exemplifica a estrutura do arquivo 'products.json', a cada novo produto é adicionado +1 em 'nextId' e adicionado mais um produto a lista de 'products'. Repare também que 'categorys' é um array, já que um produto as vezes pode pertencer a mais de uma categoria, vale ressaltar que, apesar de no exemplo 'category' está sendo associada a marca (O Boticário), o conceito de categoria é mais abrangente e pode levar

Figura 47 – Exemplo do product.json – Autoria Própria

em consideração outros aspectos do produto (a critério do usuário).

Por fim, se tratando de API, todas as solicitações foram implementadas, com as conexões de acordo com a Figura 48, finalizando o desenvolvimento do back-end.

Outro fator importante para a conexão entre a API e a aplicação foi o uso da Heroku, que é uma plataforma na nuvem que faz deploy de várias aplicações back-end seja para hospedagem, testes em produção ou escalar

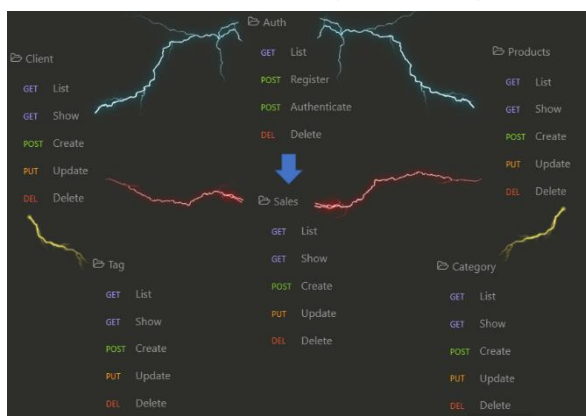


Figura 48 – Requisições API – Autoria Própria

as suas aplicações. Como a API em ‘localhost’ rodava no computador e a aplicação do expo rodava no celular a comunicação entre ambos era dificultada, com o uso da heroku e o API funcionando na nuvem se tornou mais simples, bastava trocar a baseURL para o link correspondente a API na Heroku, conforme Figura 49.

```
const api = create({
  baseURL: 'https://api-mooras.herokuapp.com',
});
```

Figura 49 – Service API do Morras linkando a baseURL ao Heroku – Autoria Própria

Sobre o front-end, além de correção de bug’s visuais, foi finalizada a página de detalhes do cliente, conforme Figura 50. Vale destacar os totais parciais no fim de cada mês, informação importante para controle de vendas. Apesar disso, a página ainda se encontra incompleta, faltando implementar a opção de edição de cliente, e edição dos valores de venda e pagamento

Gilvan	
Nasc.: 01/01/2000	Cel.: (84) 9 9876-5432
End.: Rua Teste 123, n° 70, Cond. Sol quente, Bl. 06, Ap. 103	
10/2020	
1 Coffe Masc.	165,00
pagamento (01/10) -50,00	
2 Coffe Masc.	330,00
2 Caixa sabonete.....	40,00
pagamento (25/10) -200,00	
Total até 11/2020	785,00
11/2020	
pagamento (02/11) -200,00	
pagamento (07/11) -450,00	
1 Oleo Quinoa	60,00
1 Caixa de sabonete	20,00
Total até 12/2020	215,00

Figura 50 – Página de detalhamento de cliente – Autoria Própria

2.6.2. Lições Aprendidas

Para mim, a lição que mais teve peso nesta Sprint foi compreender a importância de desenvolver a API no início da implementação. Pois é mais fácil estruturar o projeto utilizando as requisições reais desde o início. Ou compreender e estruturar muito bem os objetos antes de iniciar o desenvolvimento.

Old Tag:		New Tag:
{ key: Familia }	≠	{ id: 1, name: Familia, user: 2 }
{ key: Trabalho }		{ id: 2, name: Trabalho, user: 2 }

Figura 51 – Diferença do objeto tag – Autoria Própria

No meu caso, desenvolvi toda a interface, mockando valores e utilizando métodos que se estruturam bem nessas variáveis. Mas quando foi preciso utilizar os objetos retornados pela requisição da API foi preciso refazer a maioria dos métodos porque as estruturas dos objetos eram diferentes. Exemplo: na Figura 51, antes de fazer a conexão com a API as tag’s eram estruturadas igual a ‘Old Tag’, após o desenvolvimento da API foi necessário alterar o objeto tag, o que afetou diversos métodos que já haviam sido desenvolvimentos.

2.7. Sprint 7

A sprint 7 dará continuidade ao desenvolvimento do front-end, mais especificamente aos filtros e opções da página de cliente.

2.7.1. Solução

A seguir, os passos e resultados das etapas para o desenvolvimento do front-end do Mooras.

2.7.1.1. Evidências do Planejamento

Para registro do planejamento na Sprint, segue a captura de tela da ferramenta Trello, Figura 52, cujas tarefas realizadas se encontram na lista ao lado. É válido destacar que as tag's de cor verde significam que a tarefa foi finalizada, e cerca das tag's de cor amarela, a qual foi apresentada nesta tonalidade pois as funcionalidades ainda se encontram em desenvolvimento.

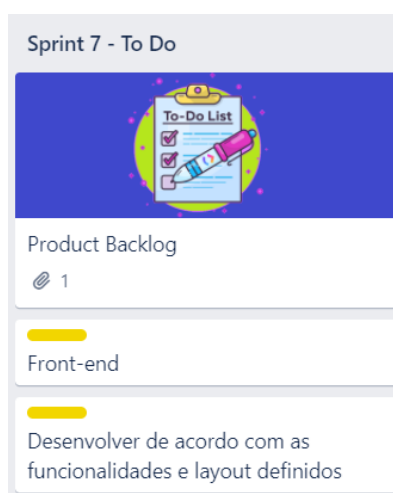


Figura 52 – Tarefas realizadas no sprint 7 – Autoria Própria

2.7.1.2. Evidência da execução de cada requisito

Para auxiliar na identificação das tarefas do front-end, realizou-se uma nova captura de tela do Trello. Na Figura 53, a cor verde em destaque tem como função representar os cards que já estão finalizados e os amarelos parcialmente completos.

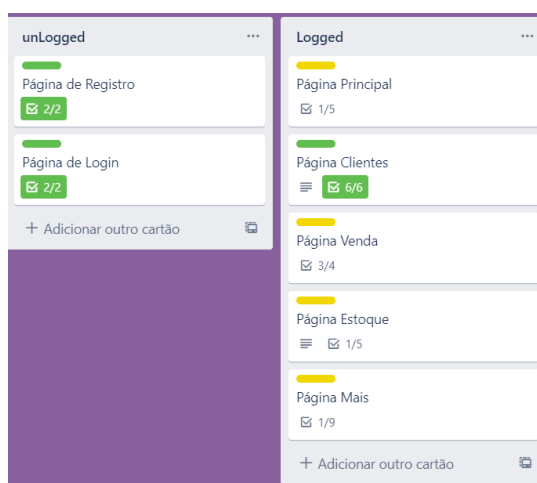


Figura 53 – Identificação das páginas completas – Autoria Própria

Comparando a Figura 53, com a Figura 43, é possível observar o grande avanço no projeto neste período de tempo. Foi finalizada a página 'Cliente' e iniciado a estruturação das demais páginas que ainda estavam em branco.

2.7.1.3. Evidência de solução

Nesta Sprint o foco foi terminar a página de cliente e assim ter um MVP, tendo todas as funcionalidades relacionadas a 'cliente' e a 'venda' desenvolvidas. Dessa forma, foi finalizado o menu, da página, Figura 54.



Figura 54 – Menu da página clientes – Autoria Própria

A funcionalidade de cada botão da esquerda para a direita: ordenação dos clientes por ordem alfabética, valor decrescente e valor crescente; busca por cliente cujas iniciais foram digitadas; filtro de tag, valor pago e valor pendente. A Figura 55 mostra o resultado do lado esquerdo da ordenação por ordem alfabética e no lado direito a ordenação por ordem do valor decrescente.

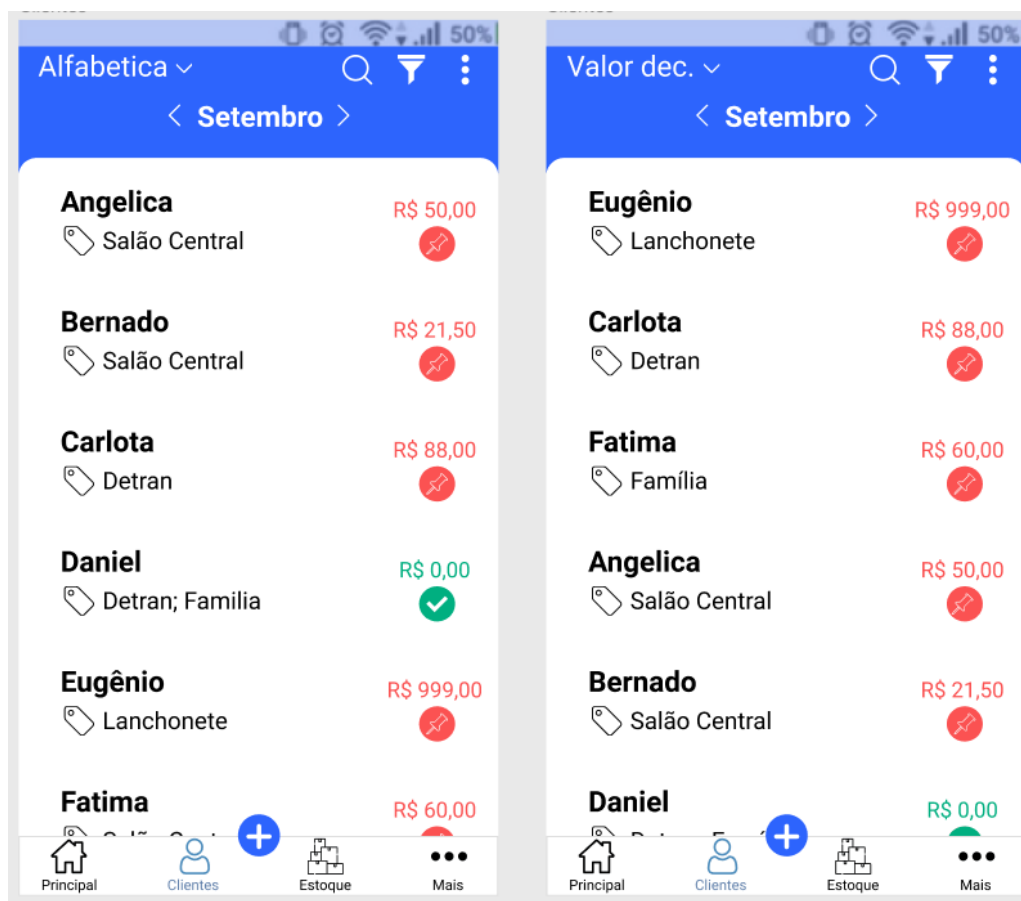


Figura 55 – Ordenação por ordem decrescente – Autoria Própria

A Figura 56 mostra o trecho de código onde é realizada a filtragem dos clientes digitados, exemplo: ao digitar no ícone da lupa 'Dan' apenas o cliente Daniel iria aparecer na tela.

```

lendo = [];
for(let i=0; i<lendoSemFiltro.length; i++){
  tam = 0;
  control = 0;
  if(lendoSemFiltro[i].titulo.length>filtro.length){
    max = filtro.length;
  }
  else{
    max = lendoSemFiltro[i].titulo.length;
  }
  for (let u=0; u<max; u++) {
    if(lendoSemFiltro[i].titulo[u].toUpperCase()==filtro[u].toUpperCase() 88 control==0){
      tam++;
    }
    else{
      control++;
    }
  }
  if(tam==max){
    lendo = [...lendo, lendoSemFiltro[i]];
  }
}

```

Figura 56 – Código de busca de sentenças semelhantes– Autoria Própria

Já na Figura 57 é possível observar o menu de filtro, é possível selecionar apenas os clientes com determinada tag ou se os clientes possuem todos os seus débitos pagos ou pendentes.

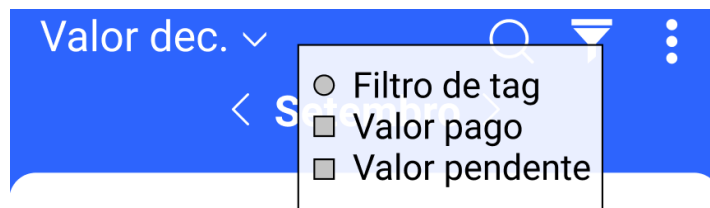


Figura 57 – Menu de filtro – Autoria Própria

Já o botão menu, o de três pontinhos, não foi desenvolvido uma funcionalidade, então será retirado na versão MVP do aplicativo.

2.7.2. Lições Aprendidas

Acredito que a maior dificuldade que passei neste projeto foi o controle de desenvolvimento. No início quando comecei minha única ocupação era a pós-graduação, então tive um grande tempo para avançar muito o projeto. Com a necessidade de se trabalhar e de me envolver em outros projetos de estudo o tempo livre para desenvolvimento foi reduzido drasticamente.

Por essa razão o projeto não foi inteiramente finalizado, porem fiz questão de finalizar completamente o modulo 'cliente' e o de 'venda', o que pode ser facilmente reproduzido para o modulo de 'estoque' (uma vez as páginas de que estoque e cliente são semelhantes). Com o que já foi desenvolvido é perfeitamente possível utilizar em campo.

Por essa ser a última Sprint, o projeto terá que ser finalizado em um trabalho futuro, restando incrementar o modulo do estoque e o dashboard dos gráficos.

3. Considerações Finais

3.1 Resultados Finais

A expectativa deste projeto era que ao final gerasse um aplicativo que pudesse gerir as vendas, os clientes e o estoque de um revendedor de cosméticos. Dois desses objetivos foram cumpridos, o controle de vendas e de clientes.

Acredito que a principal falha, e o que eu alteraria, envolve o controle de tempo e controle de desenvolvimento. Começaria o desenvolvimento pela parte do back-end e posteriormente iria para o front-end, ou faria uma análise de requisitos mais robusta que me evitasse tantos retrabalhos. Colocaria mais tempo de desenvolvimento no início do projeto, onde eu possuía mais tempo livre, e deixaria para o final apenas detalhes e melhorias. Basicamente o projeto não foi finalizado por falta de tempo.

Estou satisfeito com o framework escolhido para a implementação, o react native é muito intuitivo e possui uma documentação muito rica. Porém o professor da disciplina me alertou, que o desenvolvimento mobile possui algumas complicações a mais que o desenvolvimento web, e isso se provou verdadeiro no final. Talvez muito do tempo que perdi estudando certas situações poderiam ter sido reduzidos se eu resolvesse ter realizado um aplicativo web.

Contudo, apesar das dificuldades de implementação e de cumprimento de prazo, o resultado final me deixou satisfeito, cumprindo com perfeição, os dois principais requisitos do que era ideal para o sistema, o controle de vendas e clientes.

3.2 Contribuições

Mooras já entrou em fase de teste. Dois revendedores já o estão utilizando para armazenar os dados, histórico de compras e pagamentos de seus clientes. Nos relatos do uso, fui informado da melhora na performance do vendedor, principalmente no cadastro da venda e na busca pelo valor devedor de determinados clientes, consequentemente diminuindo o tempo ocioso do vendedor.

Além de ser um aplicativo gratuito e intuitivo comparado com outros aplicativos semelhantes, a sua interface é muito limpa, bastando ter apenas o necessário para uso do vendedor. Isso se deve principalmente ao seu foco de nicho exclusivo em revendedores de cosméticos, como outros aplicativos são mais genéricos, podem possuir algumas ferramentas que acabam se tornando não tão essenciais para algumas finalidades.

3.3 Próximos passos

O próximo passo é finalizar o aplicativo com todas as partes idealizadas e não implementadas. Uma parte essencial e que infelizmente não foi finalizada é o controle de estoque. Felizmente essa parte possui grandes semelhanças com o controle de clientes, o que pode acabar reduzindo o tempo de desenvolvimento do mesmo. Outros requisitos não funcionais é a implementação de uma tela principal 'dashboard' com todas as principais informações (como gráficos de vendas) e uma tela de ferramentas que podem ajudar o vendedor.