UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

IGOR CARNEIRO DA SILVA

POSSIBILITANDO A MANIPULAÇÃO DE COMPORTAMENTOS EM PRODUTOS VIRTUAIS SEM CONHECIMENTO DE TECNOLOGIA

RECIFE
2021

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

IGOR CARNEIRO DA SILVA

POSSIBILITANDO A MANIPULAÇÃO DE COMPORTAMENTOS EM PRODUTOS VIRTUAIS SEM CONHECIMENTO DE TECNOLOGIA

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Henrique Emanuel Mostaert Rebelo.

RECIFE
2021

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

IGOR CARNEIRO DA SILVA

POSSIBILITANDO A MANIPULAÇÃO DE COMPORTAMENTOS EM PRODUTOS VIRTUAIS SEM CONHECIMENTO DE TECNOLOGIA

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em ___ de _____ de _____.

Banca Examinadora:

_____

Prof. Henrique Emanuel Mostaert Rebelo
(Orientador)
Universidade Federal de Pernambuco

_____

Prof. Adriano Augusto de Moraes Sarmento
(Avaliador)
Universidade Federal de Pernambuco

RECIFE
2021

Dedico este trabalho aos meus pais Patrícia Ramos e Ivanilson Carneiro (in memorian), aos meus avós Samuel Carneiro, Joseane Pereira, Severina Ramos, e à minha irmã Iuly Millena.

AGRADECIMENTOS

Agradeço imensamente aos meus pais, que sempre se esforçaram para que eu tivesse um ensino de qualidade, que sempre me auxiliaram em meio a esta jornada acadêmica e da vida. Agradeço aos meus avós que também a todo momento estiveram presente buscando me ajudar, nem que fosse dando o “dinheiro do chiclete”, como eles dizem. Agradeço a todos que de alguma forma contribuíram com o meu crescimento, seja de forma financeira, seja sentando ao meu lado para estudar e tirar minhas dúvidas, seja acreditando que eu conseguiria atingir meus objetivos.

Gostaria também de dar um obrigado muito especial aos professores que fizeram parte da minha vida como tia Néia, minha querida mãe Patrícia, tia Marleide, tia Elzira, tia Vilma, tia Carmem, Vera, tia Betinha, tia Nilda, professor João, Edvaldo, Solon, Fátima, Germano, Michele, Silvana, Luciano, Marcos, Diego, Adalgisa, Edymur, Laércio, Carlos, Bruno, Miriam, Edilson Acciolly que sempre ao passar por mim perguntava se eu estava estudando e dizia para estudar mais, Geraldo Pacheco, Maciel Carneiro, Gustavo, Osglay, grande e inesquecível Pedro Paulo (Negão), Tenório, Valter Júnior, Gilberto, André Costa, Feliciano, Djair, Ozias, Macedo, Ênio, Josemar, Abelardo, Moacir, José de Lira, Remy, Sílvio Melo, Adriano Sarmento, Juliano Iyoda, Eduardo Tavares, Liliane, Henrique Rebelo, Odilon Maroja, Renato Mariz, André Luis, Otoni Nobrega, Rafael Dueiri, Paulo Gustavo, Prudêncio, Manoel Eusébio, George Darmiton, Suruagy, Manoel Lemos, Camila Michelyne, Nelson Rosa e a todos os demais professores aqui não citados mas que realizaram seu trabalho com amor e dedicação, merecendo receber o valor e reconhecimento que deveriam ter desde sempre. Sem eles a caminhada até aqui seria infinitamente mais difícil.

“Existem três tipos de tolos no mundo. O primeiro é aquele que não sabe, mas não sabe que não sabe - é o tolo simples. O segundo é aquele que não sabe, mas acha que sabe - é o tolo complexo, o tolo instruído. E o terceiro é aquele que sabe que não sabe - o tolo abençoado. ”

Osho

RESUMO

Em uma empresa, os profissionais de cada área utilizam ferramentas específicas para desenvolver suas atividades. Grande parte destas ferramentas possuem diversas funcionalidades que só aquele profissional consegue entender. Isto, além de impedir que integrantes de outras áreas participem ativamente de certas atividades, não faz com que os mesmos compreendam determinados processos importantes que ocorrem na empresa. Para se ter uma equipe engajada e bem integrada é importante que todos entendam e possam participar, mesmo que de maneira menos aprofundada, de atividades das áreas que não sejam as suas. Este projeto visa tornar transparente a todos os integrantes comportamentos específicos de seus produtos criados pela área de desenvolvimento, além de possibilitar a manipulação dos mesmos sem qualquer conhecimento tecnológico, tornando estas modificações disponíveis em produção instantaneamente.

Palavras-chave: arrastar e soltar, manipulação de comportamento, produtos virtuais

ABSTRACT

In a company, professionals in their areas use specific tools to develop their activities. Most of these tools have several features that only that professional can understand. This prevent members from other areas from actively participating in certain activities and does not make them understand certain important processes that take place in the company. In order to have an engaged and well-integrated team, it is important that everyone understands and can participate, even if less deeply, in activities in areas other than their own. This project aims to make the specific behaviors of their products created by the development area transparent to all members, in addition to enabling them to be manipulated without any technological knowledge, making these modifications instantly available in production.

Keywords: drag and drop, behavior manipulation, virtual products

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

Sigla	Significado	Página
HTML	HyperText Markup Language	16
CSS	Cascading Style Sheets	16
API	Application Programming Interface	17
REST	Representational State Transfer	26
ETS	Erlang Term Storage	30

1. Introdução

Neste capítulo serão apresentados os elementos que motivaram o desenvolvimento da ideia proposta neste trabalho, quais problemas se está tentando resolver e quais objetivos se busca atingir.

1.1. Motivação

A idealização dos produtos desenvolvidos por uma empresa pode envolver integrantes de diversas áreas, seja para definir o design, sua forma de monetização ou para qual tipo de dispositivo eletrônico eles serão criados.

Quando se trata de crescimento, qualquer contribuição é bem-vinda, e por isso é de extrema importância que a maior parte da equipe entenda, nem que seja de maneira menos aprofundada, os processos de todos os pontos da empresa. Em específico, é importante a compreensão de quais comportamentos e fluxos de seus produtos são possíveis de serem criados ou manipulados, fazendo com que qualquer pessoa da equipe produza bons insights e possa discutir por melhorias mais assertivas em suas recorrentes reuniões de trabalho. No entanto, a área responsável por dar vida aos comportamentos e fluxos dos produtos é a área de desenvolvimento, esta sendo vista muitas vezes como algo difícil de se compreender por se trabalhar com linguagens de programação e outras tecnologias.

Para que todos aqueles que não são da área de desenvolvimento possam participar ativamente da criação e manipulação de comportamentos de seus produtos virtuais, este projeto propõe uma implementação de funcionalidades específicas nos projetos backend e uma interface web responsável por definir os comportamentos de seus produtos de maneira simples, sem a necessidade de conhecimento de qualquer tipo de tecnologia, linguagem ou lógica de programação. Os comportamentos serão definidos arrastando e soltando blocos com comandos específicos, formando uma sequência de ações.

1.2. Problema

Muitas ações, que contribuem de alguma forma para o crescimento da empresa, não são compreendidas nem acessíveis para manipulação por integrantes que não compõem aquela área específica, ou seja ferramentas para design só designers entendem, ferramenta para vendas só a equipe de venda entende e ferramentas de desenvolvimento só a equipe de desenvolvimento sabe como utilizar. Com isso, uma alternativa é contratar mais pessoas, o que aumenta o trabalho de gerenciamento de equipe e também aumenta os gastos da empresa, já que este novo integrante deverá receber por suas atividades.

Outras plataformas que buscam facilitar o desenvolvimento de produtos manipulando blocos ou são pagas ou possuem funcionalidades limitadas, além de serem desenvolvidas para programadores, ou seja, requer um conhecimento de lógica computacional, e para cada alteração realizada é necessário atualizar o projeto em produção para que o novo comportamento fique acessível. Já a forma aqui proposta, além de ser gratuita (já que será algo desenvolvido pela própria empresa), será de fácil acesso e manipulação, permitindo que a empresa utilize a linguagem de programação que achar melhor para o desenvolvimento de seus produtos, além de tornar a atualização disponível em produção instantaneamente, não precisando realizar um novo deploy.

Uma comparação entre algumas plataformas já existentes e a ideia apresentada neste trabalho é realizada no tópico 5.2 (Comparação com as plataformas existentes).

1.3. Objetivos

Este trabalho tem como objetivo aumentar a bagagem de conhecimento de todos da empresa em relação aos possíveis comportamentos que seus produtos virtuais podem ter. Além disso, através de uma interface web simples e direta, tornar possível a manipulação destes comportamentos sem que se tenha um entendimento aprofundado das tecnologias utilizadas. Para que isso aconteça, precisamos ter uma aplicação backend capaz de armazenar estes comportamentos.

- Objetivos gerais:
    - Tornar possível a criação e manipulação de comportamentos sem que se tenha conhecimento sobre tecnologia.

    - Tornar transparente certos comportamentos que os produtos virtuais da empresa possuem.

- Objetivos específicos:
    - Definir a estrutura de um projeto backend, utilizando a linguagem de programação Elixir e o framework Phoenix, para possibilitar a manipulação e execução de sequências de funções.
    - Ter o melhor desempenho possível ao executar operações com grande quantidade de iterações acessando o banco de dados.
    - Ter uma interface web para criar e manipular os comportamentos dos produtos virtuais de forma simples, arrastando e soltando blocos.

2. Conceitos Básicos

Neste capítulo será apresentado o que um produto precisa conter para ser considerado um produto virtual e o que se quer dizer com manipulação de comportamentos.

2.1. Produtos Virtuais

Neste trabalho, considera-se produtos virtuais todos aqueles softwares desenvolvidos pelas empresas que serão utilizados através de computadores, celulares, relógios ou quaisquer outros dispositivos que consigam executar o software. De maneira geral, se o produto foi construído para se comportar de maneira dinâmica, ele se encaixa na categoria aqui descrita de produtos virtuais, que estaremos dando foco neste trabalho, como aplicativos, sites e até aplicações backend, que não são necessariamente acessadas diretamente pelo usuário final.

Para deixar claro o que se está querendo dizer com aplicações que possam alterar seus comportamentos de forma dinâmica, precisamos entender de quê as aplicações são constituídas. As aplicações web (e-commerces, blogs, redes sociais, etc...) por exemplo, podem ser construídas por linguagens de marcação (como HTML), uma definição de estilos (por exemplo, usando CSS) e uma linguagem de programação para definir seus comportamentos (como o JavaScript). Se uma aplicação é construída apenas com a

linguagem de marcação e/ou a definição de estilo, não temos aqui um tipo de produto virtual que será tratado neste trabalho. Mesmo que a definição de estilo tenha o poder de alterar o design da página dependendo das dimensões da tela onde o software está sendo executado, não é o suficiente para se ter um controle mais rebuscado em relação ao que aquela aplicação pode fazer. Temos assim, uma aplicação estática. Uma página web apenas para divulgação de algum produto de beleza ou uma aplicação mobile apenas para consultar informações da tabela periódica são exemplos de aplicações estáticas. Quando adicionamos a linguagem de programação, passamos a ter a ferramenta responsável por definir seus comportamentos a ponto de podermos alterá-los de maneira simples dinamicamente, e agora passamos a ter um tipo de produto frontend do qual se trata neste trabalho. Outro ponto importante a se considerar é que estas aplicações frontend, além de terem a linguagem de programação para definir seus comportamentos, devem estar acessando alguma API (também desenvolvida pela mesma empresa), pois é dela que virá quais comportamentos aquelas aplicações devem ter. Dado que as aplicações backend são construídas para realizarem um conjunto de comportamentos específicos, todas estas aplicações se encaixam no grupo de produtos virtuais abordados aqui. Então, no geral, sempre que for lido ‘produtos virtuais’ deve-se pensar em aplicações frontend que estão consumindo informações de alguma aplicação backend, ou as próprias aplicações backend também construídas por aquela empresa.

2.2. Manipulação de Comportamentos

Todas as aplicações frontend que não são estáticas possuem comportamentos, seja mostrar uma notificação quando uma cliente clica em algum botão, seja redirecionar para uma página específica, ou até bloquear ou não o acesso de alguns conteúdos a certos tipos de usuários. Para as aplicações backend temos por exemplo, montar uma mensagem de instruções e enviá-la para a cliente por e-mail ou pelo whatsapp. Estes são alguns exemplos de comportamentos que iremos tornar disponíveis sua manipulação, podendo alterá-los e até criar uma combinação nova de ações, construindo assim um novo comportamento.

Para tornar possível esta manipulação, precisamos deixar claro alguns conceitos. Primeiro devemos entender o que se quer dizer com a palavra ‘função’. Em uma linguagem de programação, uma função é um bloco que agrupa um conjunto de

comandos, podendo ou não receber informações como dados de entrada e da mesma forma podendo ou não retornar alguma resposta para o ponto específico do código que a chamou. Segundo, entender que quando se fala em comportamento se está querendo dizer uma ou mais funções executadas em sequência.

O que vamos modificar ou criar são os comportamentos (sequências de funções), e não as funções em si. Para que isso seja possível, devemos ter ao final uma estrutura criada no backend capaz de armazenar uma lista de informações das funções implementadas. Após isso, precisamos implementar uma forma de executar as funções a partir destas listas, assim realizando os comportamentos desejados.

3. Trabalhos Relacionados

Aqui serão apresentadas plataformas semelhantes, como OutSystems, Kodular, Thunkable e Bubble. Estas plataformas funcionam arrastando e soltando componentes, e são utilizadas para o desenvolvimento de aplicações do zero, se assemelhando em relação ao manuseio e ao fato de serem utilizados para desenvolvimento de produtos virtuais.

3.1. OutSystems

OutSystems é uma plataforma que fornece soluções de desenvolvimento low-code(com pouco código) para que as empresas possam automatizar processos e criar soluções internas de forma mais rápida do que com programação tradicional. Esta plataforma possui uma interface que representa cada parte do código de uma forma visual, mas mesmo assim há várias áreas em que se precisa saber trabalhar com códigos, sendo uma ferramenta criada para programadores.

Com o OutSystem é possível criar aplicações web, aplicações mobile nativas e fazer integrações com APIs, mas não possui um ambiente que possibilite o desenvolvimento sem manipulação de código tradicional. Sua curva de aprendizado é alta e por ser uma plataforma muito cara apenas empresas de grande porte a utilizam.

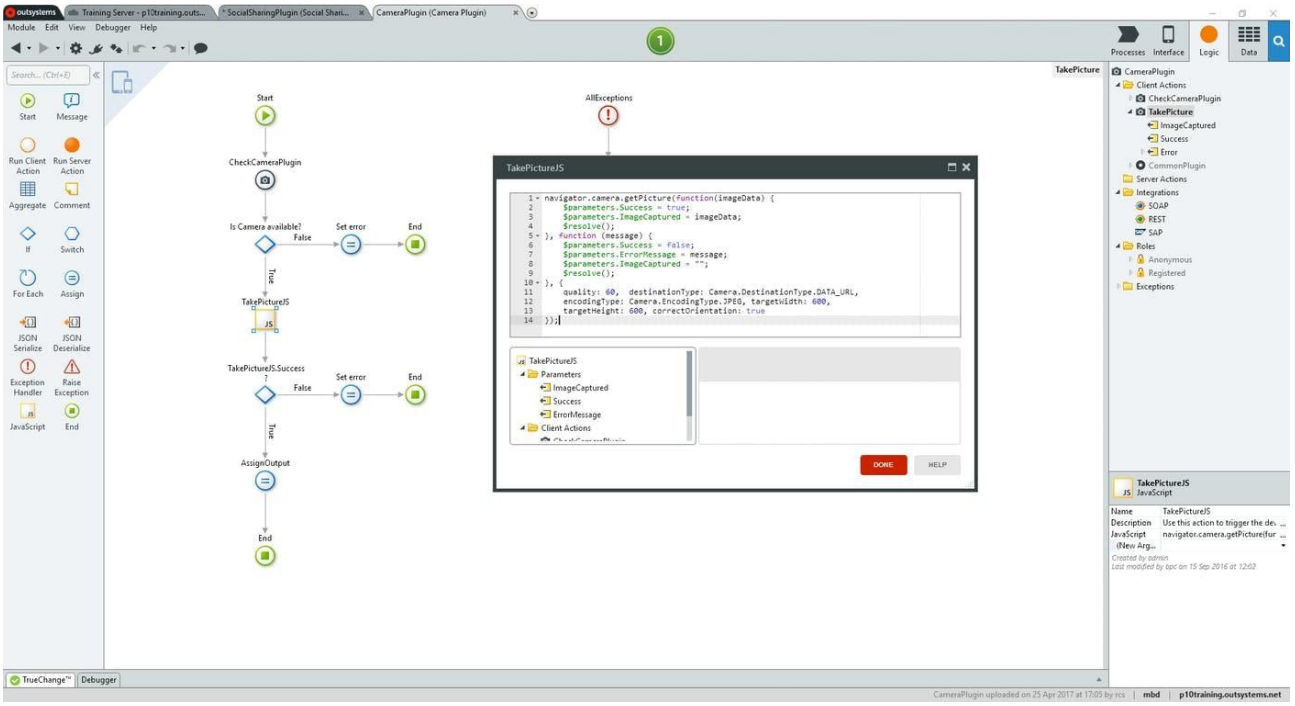

Figura 01: Interface da plataforma OutSystem com blocos de ações e trechos de código.

3.2. Kodular

Plataforma de desenvolvimento mobile, sendo possível apenas desenvolvimento para o sistema operacional android.

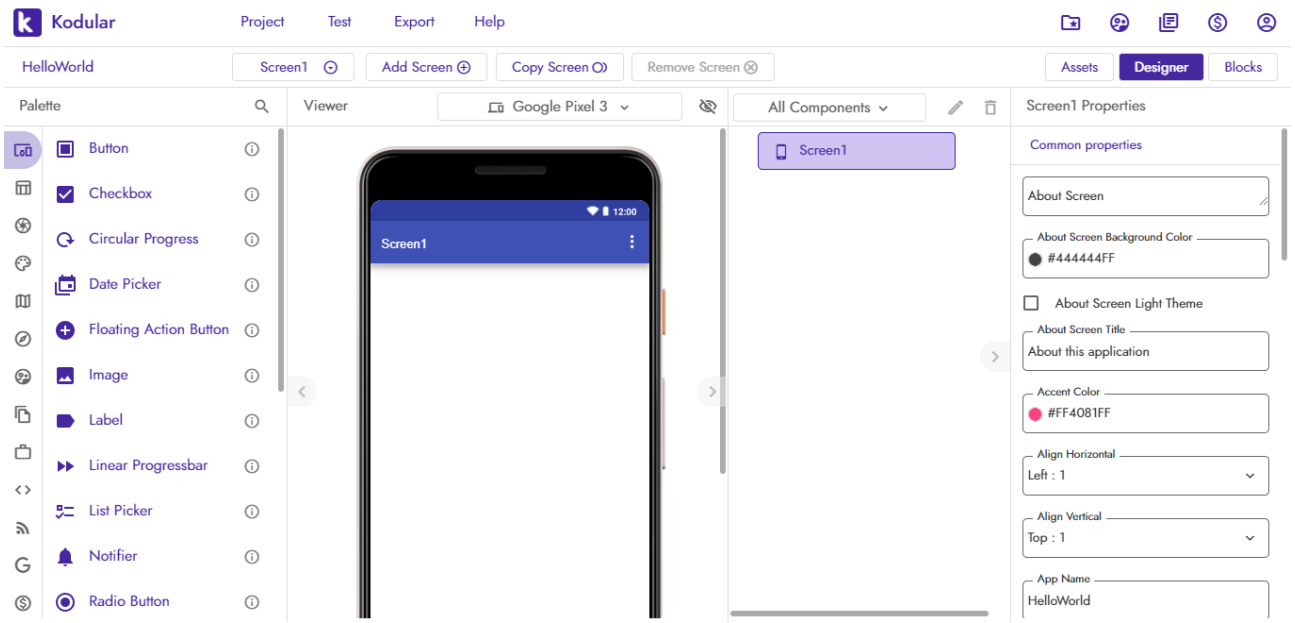

Figura 02: Tela de criação de aplicações da plataforma Kodular.

Os comportamentos da aplicação são definidos por blocos, mas mesmo assim requer de quem for manipular, conhecimentos de lógica de programação e algoritmos. Seu modelo de precificação é bastante simples, sendo uma plataforma gratuita independente de quantas aplicações se crie, sendo cobrada uma taxa apenas quando há anúncios. Um dos problemas da plataforma Kodular é que qualquer aplicação desenvolvida fica inutilizável caso o sistema saia do ar. Há a possibilidade de importar o projeto em outra plataforma, neste caso podendo surgir diversos problemas relacionados com compatibilidade.

3.3. Thunkable

O Thunkable é uma plataforma no-code (sem utilização de código tradicional) utilizado para desenvolvimento de aplicações mobile, tanto para o sistema operacional android quanto para o iOS. Mesmo sendo utilizado para desenvolvimento cross-plataform, ou seja um único código-fonte é compilado em código nativo dos dois sistemas operacionais citados, para desenvolver para iOS temos menos componentes que para android.

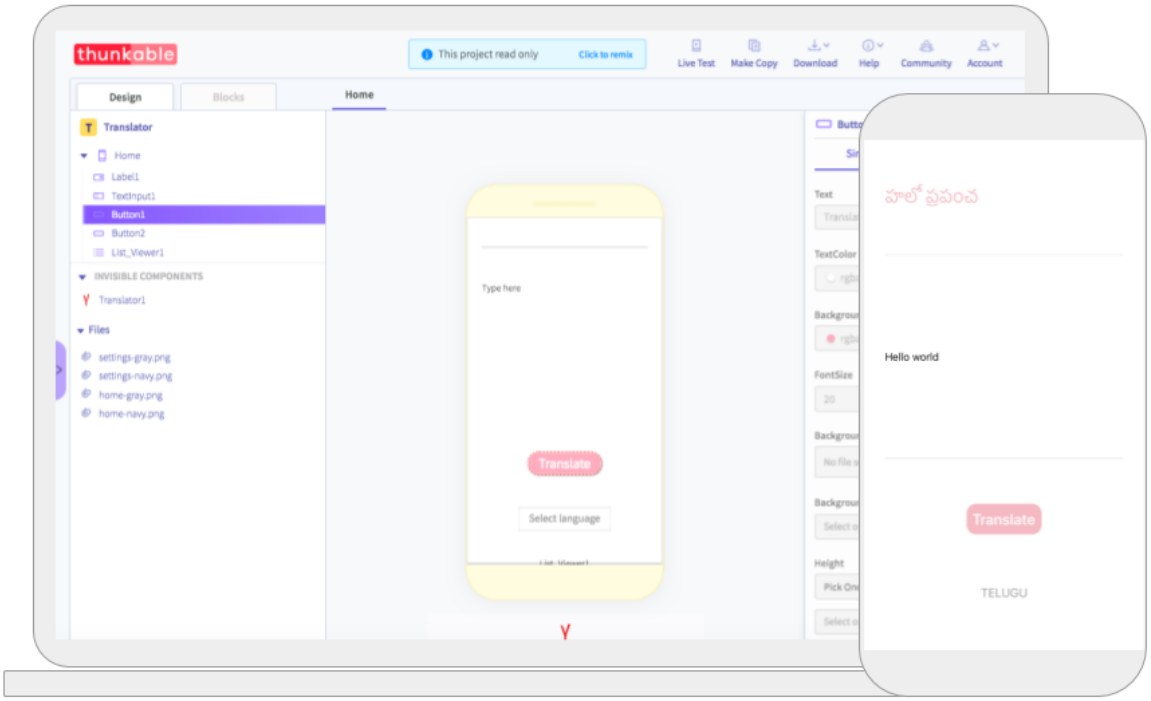

Figura 03: Tela de criação de aplicações da plataforma Thunkable.

Algo bastante interessante no Thunkable é a possibilidade de criar o que a plataforma chama de Remix, que é utilizar uma aplicação já existente, criada por algum integrante de sua comunidade, para criar uma aplicação nova adicionando, removendo ou até mudando os componentes já existentes de lugar.Esta plataforma possui um plano gratuito, contudo as aplicações desenvolvidas ficam disponíveis em sua comunidade para que outros desenvolvedores possam acessá-las. Caso se queira utilizar a aplicação desenvolvida de forma comercial é necessário assinar o plano pago.

3.4. Bubble

O Bubble é uma ferramenta no-code que permite o desenvolvimento de aplicações web e mobile (não criando aplicativos de forma nativa) de forma visual, onde os comportamentos são definidos por blocos, que te dão a opção por exemplo “ao clicar nesse botão você vai para a página de agradecimentos e envia um e-mail”. É uma plataforma que inclui muitas páginas e opções de modificação, com uma curva de aprendizado média.

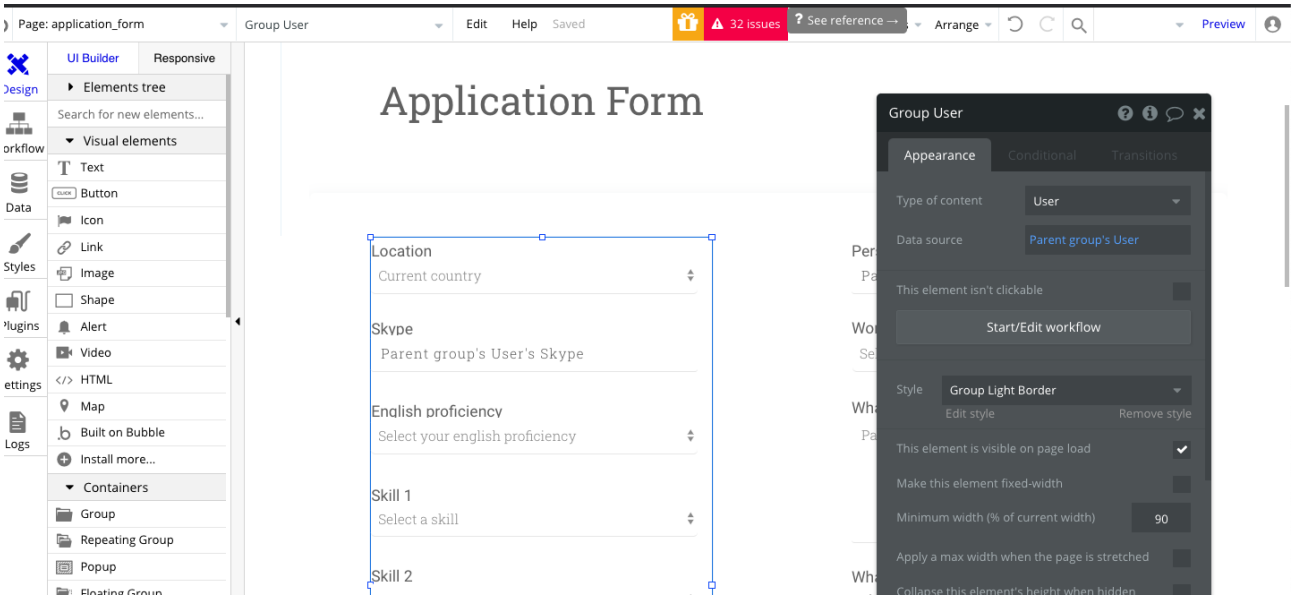

Figura 04: Tela de criação de aplicações da plataforma Bubble.

Para se criar uma boa aplicação é importante também se ter um conhecimento básico de design para que os componentes se ajustem de forma adequada quando se

trata de responsividade. É uma ferramenta desenvolvida para programadores e para comercializar sua aplicação é necessário mudar para o plano pago. No plano não pago algumas ações são limitadas. Para mais de um desenvolvedor trabalhar utilizando o Bubble precisa pagar mais, subir de plano.

4. Estrutura e Funcionamento

Neste capítulo serão mostradas em detalhes, quais informações se devem armazenar na aplicação backend para possibilitar a manipulação de sequência de funções, explicitando quais rotas serão utilizadas para realizar a inserção, atualização e deleção destas informações. Como todos estes dados ficarão armazenados no banco de dados, será apresentado também uma forma de otimizar estas operações, minimizando os tempos de acesso. Por fim, será apresentada a tela final da interface web implementada para criar e manipular comportamentos, detalhando os funcionamentos de seus componentes.

4.1. API para Manipulação de Comportamentos

Neste projeto, estaremos utilizando a linguagem de programação Elixir [3] para implementar a API de manipulação de comportamentos. Elixir é uma linguagem de programação funcional e brasileira, criada por José Valim em conjunto com a empresa de consultoria de software Plataformatec. Além de trazer as características das linguagens funcionais, o Elixir pode ser visto como uma nova roupagem da linguagem de programação Erlang [4], trazendo benefícios como velocidade, concorrência e alta disponibilidade. Como framework está sendo utilizado o Phoenix [5], dispondo de alta performance capaz de suportar milhões de conexões simultâneas em uma única máquina. Lembrando que, dependendo da aplicação a ser desenvolvida pela empresa, não é necessário uma linguagem de programação com todo esse poder. Como neste trabalho estaremos vendo apenas como tornar possível a manipulação de comportamentos de produtos virtuais, poderia estar utilizando qualquer outra linguagem.

Em primeiro lugar precisamos entender quais componentes esta API deve conter, referente apenas à questão de manipulação e criação de comportamentos. Os componentes mais simples são as funções. Como dito anteriormente, as funções são blocos que agrupam comandos específicos para se realizar uma determinada tarefa. Em Elixir elas podem ser desta forma:

```
7   def remove_blank_spaces(str) do
8     String.replace(str, " ", "")
9   end
```

Figura 05: Exemplo de função em Elixir.

É necessário também montar uma tabela, a qual será criada no banco de dados, para armazenar informações importantes das funções, que serão utilizadas tanto para informar aos usuários na interface web o que aquela função faz quanto para saber como acessar aquela função na API. Para exemplificar, utilizando a função da **Figura 05**, temos as colunas a seguir:

- **function**: onde devemos ter o nome da função, escrito exatamente como ela foi declarada no código. (remove_blank_spaces)
- **label**: nome da função que será mostrado na interface web, um nome menos técnico e que qualquer pessoa poderá entender. (Remover espaços)
- **description**: descrição de qual atividade aquela função realiza. (Remove os espaços em branco de um texto)
- **arity**: quantos argumentos de entrada a função recebe. Importante para validar se os argumentos passados são suficientes para executá-la e até para executar a sequência de funções, que será explicada mais adiante. (1)
- **argumentsType** e **responsesType**: indicam os tipos de dados de entrada e saída das funções, importantes também para verificar se a sequência de funções está bem formada ou não. Por exemplo, se a função em execução retorna um dado do tipo *string* e a função a seguir da sequência espera um dado do tipo *integer* esta sequência não é válida. (neste caso argumentsType e responsesType são a lista [“string”])

Além da tabela de função devemos ter a tabela de módulos. Cada módulo armazenará um conjunto de funções que realizam atividades referentes a um mesmo assunto, como por exemplo formatação de texto, conversões de moedas, conversões de temperaturas e assim por diante. Como exemplo de um módulo em Elixir, temos o seguinte:

```elixir
defmodule WorkflowApi.Context.Text do

  def upper_case(str) do
    String.upcase(str)
  end

  def remove_blank_spaces(str) do
    String.replace(str, " ", "")
  end
end
```

Figura 06: Exemplo de módulo em Elixir.

A estrutura da tabela de módulos, utilizando como exemplo o módulo acima, contém os seguintes dados:

- **module**: nome do módulo, exatamente como foi declarado no código. (WorkflowApi.Context.Text)
- **label**: nome do módulo que será mostrado na interface web, o que será de fácil entendimento para qualquer integrante da equipe. (Texto)
- **description**: descrição de quais tipos de atividades aquele módulo agrupa. (Manipula textos)

A última tabela a ser criada é a estrutura de sequências de funções, esta com o intuito de armazenar uma lista identificadores de funções, definindo assim um comportamento. Esta estrutura tem as seguintes informações:

- **name**: nome da sequência, que será utilizada para sua busca no banco de dados e sua execução.
- **description**: descrição informando o que aquela sequência realiza.
- **functions_sequence**: lista de identificadores das funções que aquela sequência armazena.

Após a definição de quais informações vamos armazenar em cada tabela devemos criá-las no banco de dados. O framework Phoenix já vem com o banco de dados Postgres integrado e é ele que está sendo utilizado neste projeto. No caso do Elixir, devemos criar também os schemas [6] que servem para mapear as informações contidas no banco de dados para uma estrutura de dados.

O próximo passo é estabelecer quais rotas serão utilizadas para implementar as funcionalidades desejadas. Em relação às funções teremos as seguintes rotas:

Rota	Método	Descrição
/function/create	POST	Cria as estruturas das funções
/function/list	GET	Lista todas as funções existentes
/function/:id	GET	Busca uma função específica
/function/update/:id	PUT	Altera informações de uma função específica
/function/delete/:id	DELETE	Exclui uma função do banco de dados

Tabela 01: Rotas para manipulação de funções.

Para se trabalhar com os módulos temos as rotas a seguir:

Rota	Método	Descrição
/module/create	POST	Cria as estruturas dos módulos

/module/list	GET	Lista todas os módulos existentes
/module/update-functions/:id	PUT	Adiciona ou remove funções em um módulo
/module/delete/:id	DELETE	Exclui aquele módulo do banco de dados

Tabela 02: Rotas para manipulação de módulos.

Já em relação a estrutura de sequências teremos algo semelhante ao que fizemos com a estrutura de funções, apenas adicionando um POST descrito na tabela abaixo:

Rota	Método	Descrição
/sequence/execute	POST	Recebe parâmetros e executa a sequência de funções desejada

Tabela 03: Rota adicional, responsável por executar uma sequência.

Por fim, podemos enxergar melhor os componentes criados até então com a imagem da **Figura 07**, mostrando que temos funções (representadas por triângulos) onde cada cor representa um tipo diferente (por exemplo, funções de manipulação de textos em laranja, funções aritméticas em azul, etc), temos módulos que agrupam funções do mesmo tipo e sequências que agrupam qualquer tipo de função.

O que temos de diferente da maioria dos arquivos de rotas que podemos encontrar nas implementações de REST APIs (interfaces para definição de protocolos usados no desenvolvimento e integração de aplicações) é a existência desta última rota, que foi citada na **Tabela 03**, que executa uma sequência de funções e é responsável por realizar os comportamentos que iremos criar através da interface web.

Levando em consideração que já temos algumas funções criadas e alguma sequência inserida no banco de dados, para executá-la precisamos passar apenas o nome da sequência e os parâmetros desejados como parâmetros desta rota.

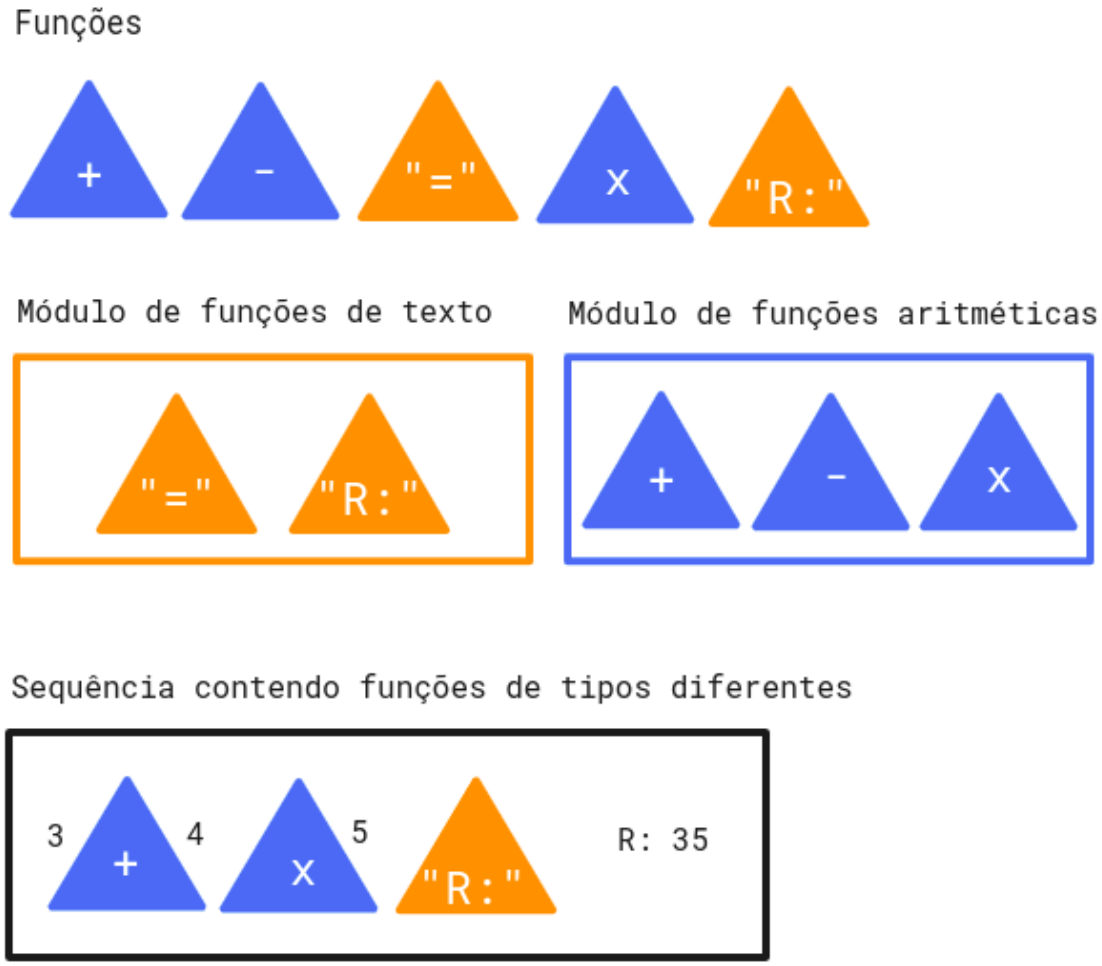

Figura 07: Mostrando como o armazenamento de funções, módulos e sequências podem ser vistos.

Para deixar mais claro, utilizando a sequência da imagem da anterior, podemos ter uma sequência contendo, nesta mesma ordem, uma função que soma dois números, em seguida multiplica o resultado da operação anterior por 5 e por fim, coloca o resultado no final do texto “R: “. Para esta sequência daremos o nome de “sequência 1”. Sendo assim, para a rota de execução de sequências podemos passar o seguinte corpo na requisição:

{

“**name**”: “sequência 1”,

“**params**”: [3, 4, 5]

}

Passamos apenas três elementos na lista de parâmetros porque as únicas funções a consumir os dados de entrada serão a primeira e a segunda, neste caso a função de soma e multiplicação, que precisam apenas de três números no total. As funções (exceto a primeira da sequência ou aquelas que sucedem uma função sem retorno) terão como primeiro argumento de entrada o resultado da função anterior.

Dado que temos como informações o nome da sequência a ser executada e os parâmetros que esta sequência irá consumir, podemos detalhar o passo a passo de como ela funciona (ler Apêndice B - *Pseudo-código e código em Elixir da execução de uma sequência*, para mais detalhes do funcionamento). A primeira ação é buscar esta sequência, a partir de seu nome, no banco de dados. Como cada item da sequência contém o identificador da função alocada para aquela posição, podemos montar uma lista de funções.

É importante lembrar que as funções podem ser removidas do banco de dados, sendo assim ao montar a lista de funções precisamos verificar se algum elemento possui o valor null (nil em Elixir). Caso alguma função desta sequência não exista mais, sua execução deve retornar um erro. Já no caso de todas as funções existirem podemos prosseguir para o próximo passo. Dado que temos agora todas as funções necessárias calculamos a aridade da sequência. Não devemos concluir que a aridade da sequência é a soma das aridades das funções. Como as funções podem receber como primeiro argumento de entrada o resultado da função anterior e também há funções que não retornam resultado, o cálculo da aridade da sequência se torna um pouco mais elaborado (para mais detalhes ler Apêndice A - *Pseudo-código e código em Elixir do cálculo de aridade de uma sequência*). Para exemplificar este cálculo vamos tomar como base a imagem da **Figura 08.** Esta figura contém três funções diferentes, cada uma com sua aridade e apenas a segunda função não retornando resposta após sua execução. Quando a sequência for executada a segunda função terá como primeiro argumento a resposta da primeira função, sendo necessário apenas que o usuário forneça o número do whatsapp.

Como a segunda função não retorna resposta, o usuário precisa fornecer todos os dados que a terceira função irá consumir. Com isso, dos argumentos fornecidos pelo usuário, a primeira função consome um, a segunda um e a terceira função consome três, fazendo com que a sequência tenha aridade 1 + 1 + 3 = 5, diferente da soma das aridades das funções, que resultaria em 1 + 2 + 3 = 6.

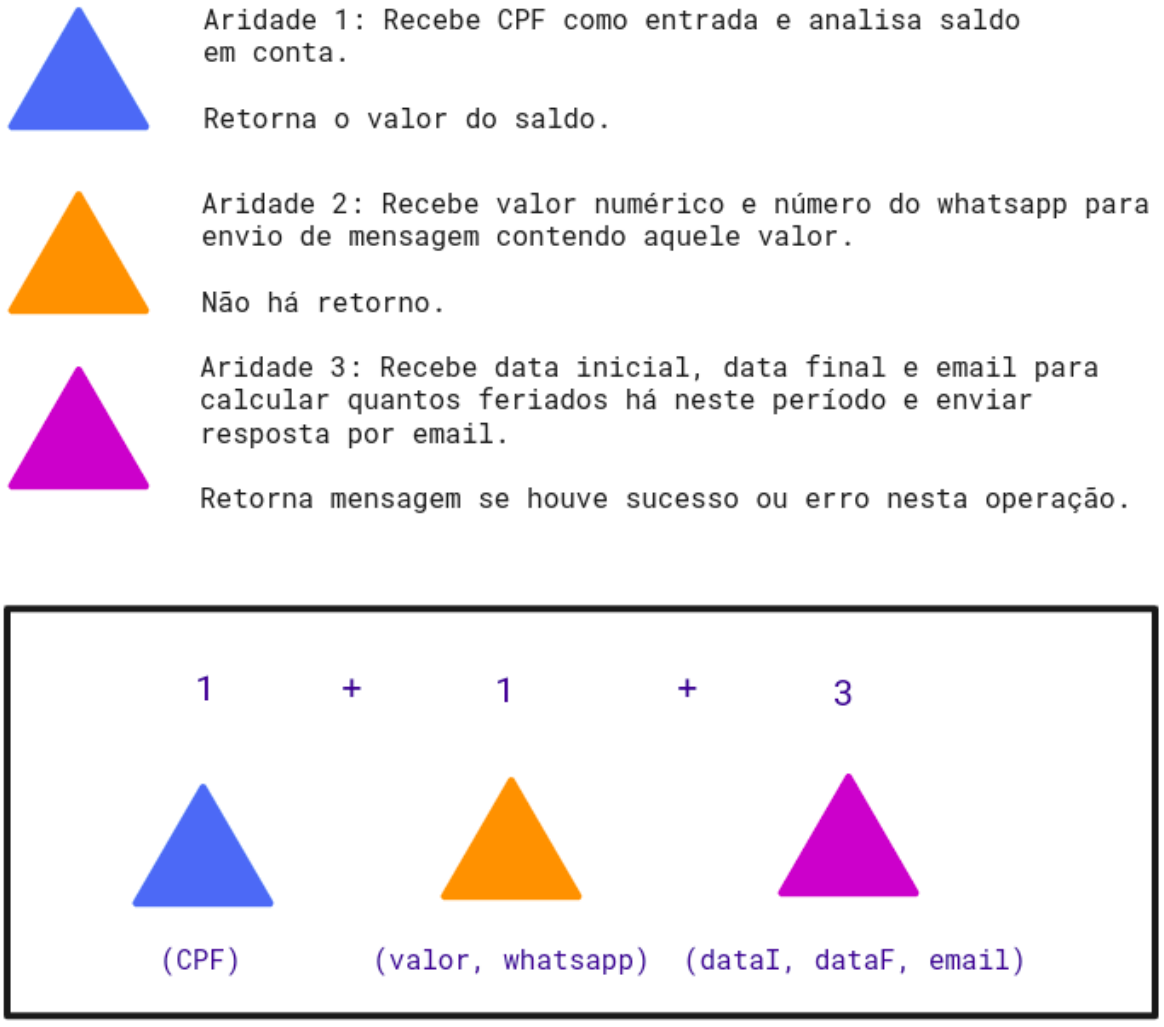

Figura 08: Exemplo de cálculo de aridade de uma sequência.

Como exemplo de um possível corpo a ser enviado na requisição de execução de sequência contendo esta sequência da **Figura 08**, temos:

{

“**name**”: “saldo e feriados”,

“**params**”: [“234.151.664-98”, “(81) 9 9634-6260”, “01/03/21”, “01/05/21”, “example@mail.com”]

}

Sabendo a aridade da sequência comparamos com a quantidade de parâmetros recebida. Estes dois valores devem ser iguais, caso contrário a computação da sequência não será realizada.

4.2. Otimização em relação ao desempenho

Sempre que uma sequência precisar ser executada ela deverá ser buscada no banco de dados junto com a lista de funções que contém, mas acessar o banco de dados é uma operação muito custosa.

Para evitar o acesso excessivo ao banco de dados iremos, a cada sequência acessada, salvá-la em memória. É importante destacar que para cada sequência editada precisamos fazer o mesmo, para deixar a memória atualizada com as modificações realizadas. No caso do Elixir estaremos utilizando o Erlang Term Storage (ETS) [14] que é um mecanismo de armazenamento de tabelas em memória do Erlang. Como o Elixir pode ser visto como uma roupagem nova da linguagem Erlang e é executado sobre a máquina virtual da mesma, então se consegue acessar alguns recursos que a Erlang disponibiliza. Com o ETS podemos armazenar uma grande quantidade de dados e manter o tempo de acesso constante.

Para testar o desempenho com e sem ETS iremos simular quantidades variadas de clientes acessando milhares de vezes a API. Estaremos trabalhando com 1, 2, 5 e 10 clientes realizando 2 mil, 3 mil, 6 mil e 11 mil requisições. Para cada simulação o tempo de resposta será contabilizado, sendo descartados os tempos das 1 mil primeiras requisições, que para efeito de análise de desempenho é o período em que o software ainda está se adaptando e realizando suas otimizações. As medições foram realizadas com apenas a API aqui desenvolvida em execução e uma aplicação Excel para colocar os dados de tempos de resposta coletados. Informações referentes à máquina em que as medições foram realizadas são mostradas na tabela a seguir:

Dados da máquina na execução	Valor
Processador	Intel® Core™ i5-10210U CPU @ 1.60GHz × 8
Memória	7,6 GiB
Sistema Operacional	Ubuntu 20.04.1 LTS
Capacidade de Armazenamento	256,1 GB
Tipo	64 bits
Bluetooth ligado	Não
Internet ligada	Não

Tabela 04: Informações a respeito da máquina utilizada para análise.

Na vida real os clientes e o servidor não estarão sendo executados na mesma máquina. Para isso iremos trabalhar com um mecanismo do Elixir chamado Nó. Cada nó executa todos os seus processos de forma isolada, criando uma simulação ideal de cliente e servidor. No terminal, ao executar a linha de comando *iex --sname client -S mix* estaremos criando um nó para o cliente. Para criar o nó do servidor abrimos um outro terminal e executamos *iex --sname server -S mix*. Ao criar o servidor teremos como resposta o nome do nó, que varia de acordo com a máquina. Para o computador que foi utilizado o nome definido foi *server@igor-Inspiron-5590.* Para que estes dois comandos funcionem como esperado é necessário estar dentro da pasta do projeto da API. Após a criação dos nós, precisamos conectá-los, executando a linha de comando *Node.connect :"server@igor-Inspiron-5590"* no nó do cliente. Feito isso, as funções podem ser chamadas a partir do cliente e o servidor responderá.

A seguir temos as tabelas dos tempos médios de resposta com e sem ETS, onde cada grupo de barras refere-se a uma quantidade de clientes (indicado na parte de baixo do gráfico) e as cores se referem à quantidade de requisições analisadas.

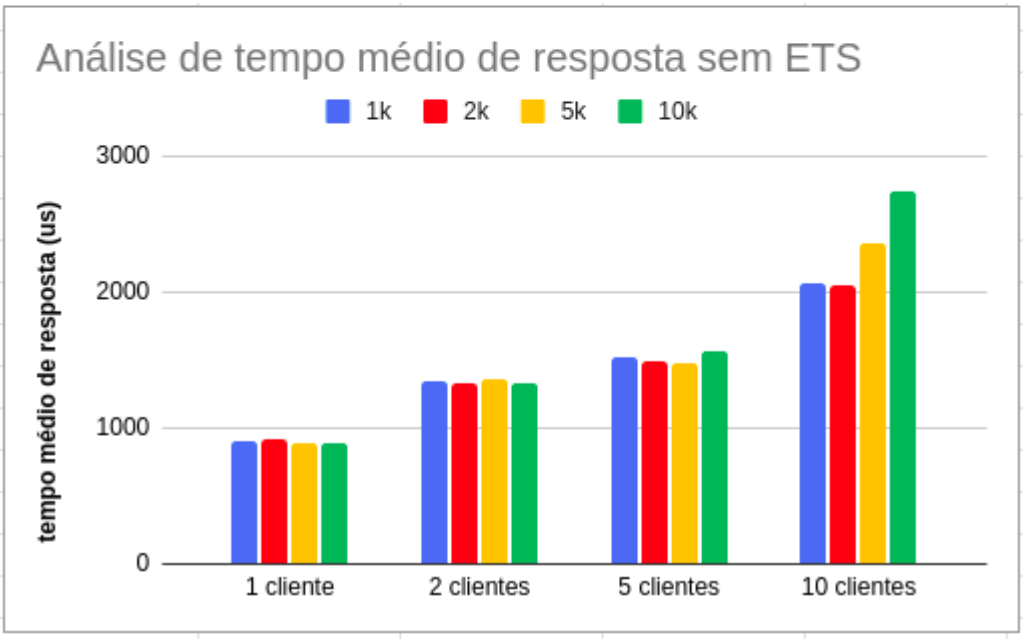

Figura 09: Gráfico dos tempos médios de respostas sem ETS.

Nota-se que para os grupos de 1, 2 e 5 clientes aumentar o número de requisições não influencia no tempo médio de resposta. Já para 10 clientes verifica-se um aumento considerável quando analisadas 5 mil e 10 mil requisições.

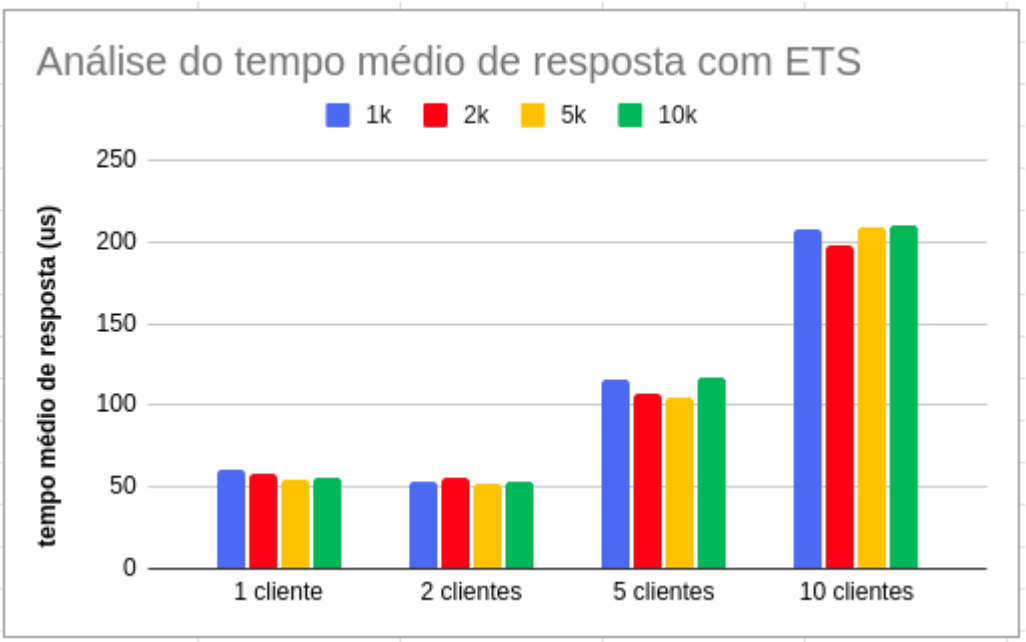

Figura 10: Gráfico dos tempos médios de respostas com ETS.

Já utilizando a otimização com ETS, como mostrado na **Figura 10**, notamos que para cada grupo o tempo médio de resposta se mantém praticamente constante, dado que a unidade de tempo está em microssegundos. É importante notar que o tempo médio de resposta cai drasticamente neste segundo gráfico. Tomando o grupo de 1 cliente como referência verifica-se que sem a otimização o tempo médio de resposta fica próximo a 1 milissegundo. Já com a otimização temos uma queda para próximo de 50 microssegundos, ou seja 20 vezes mais rápido.

4.3. Interface web para Manipulação de Comportamentos

Com a API funcionando corretamente, as sequências já podem ser criadas e manipuladas, mas sem uma interface este trabalho se torna difícil de ser feito por qualquer integrante da equipe. Por isso, esta interface vem com o objetivo de fazer com que qualquer pessoa, mesmo sem o conhecimento de tecnologia, consiga gerenciar os comportamentos de seus produtos virtuais com facilidade. O framework utilizado para a construção deste projeto foi o Vue.js [7], mas assim como a API a escolha fica a cargo da equipe. Independente do framework esta interface pode ser construída com suas devidas funcionalidades.

A imagem a seguir mostra o resultado final da tela que os usuários irão interagir:

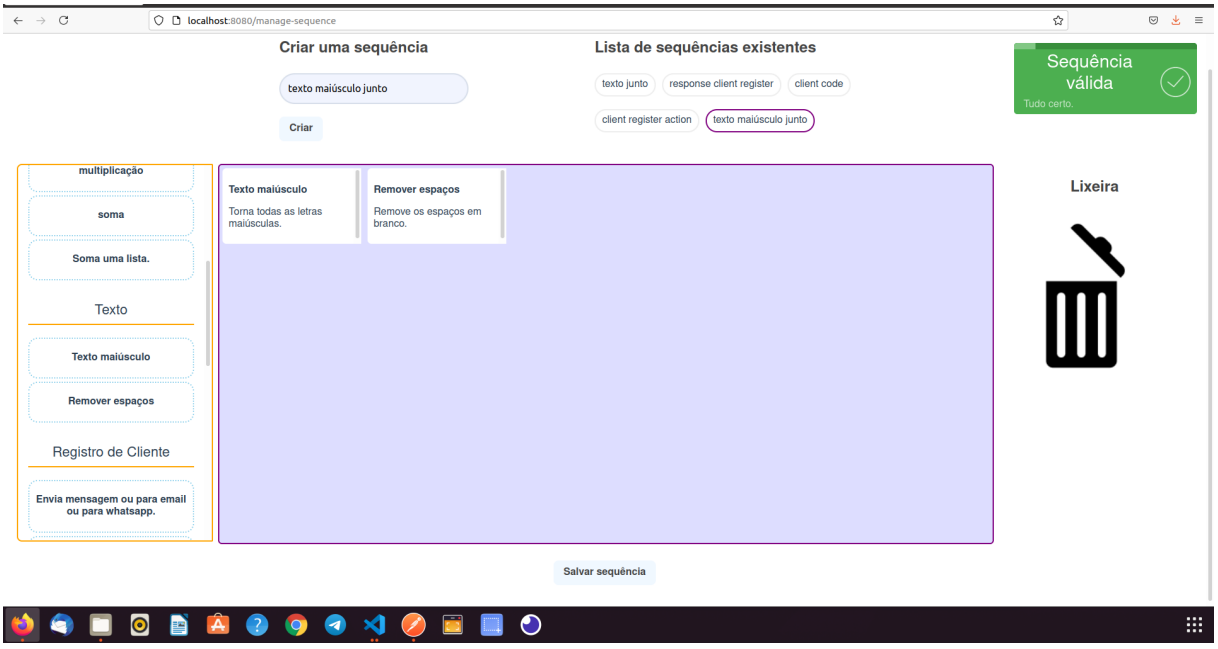

Figura 11: Tela completa da interface web para manipulação de comportamentos.

Podemos dividir a imagem anterior em alguns pontos para melhor visualização e entendimento.

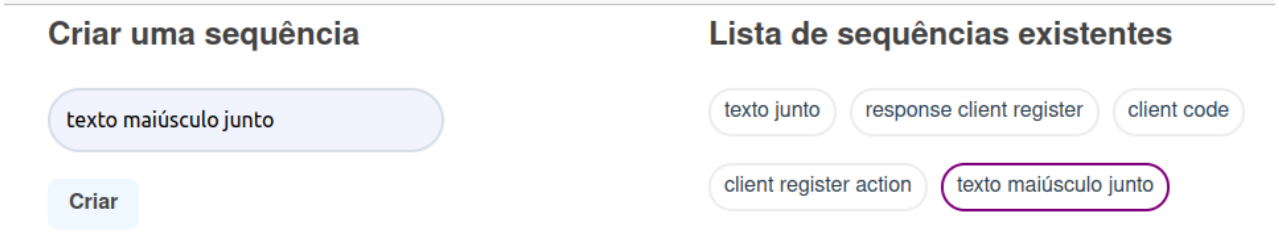

Figura 12: Componentes para criação de sequência (esquerda) e lista de sequências existentes (direita).

Na **Figura 12** temos a área de criação da sequência (esquerda) onde será definido seu nome. Ao clicar no botão 'Criar' uma sequência vazia será criada e imediatamente será listada ao lado, na lista de sequências. Já ao lado direito, temos a lista de todas as sequências existentes, ou seja aquelas que já estavam armazenadas no banco de dados junto com as que estão sendo criadas naquele momento. Ao clicar em um destes botões da lista, a sequência é exibida na área de sequências, como mostra a **Figura 13**.

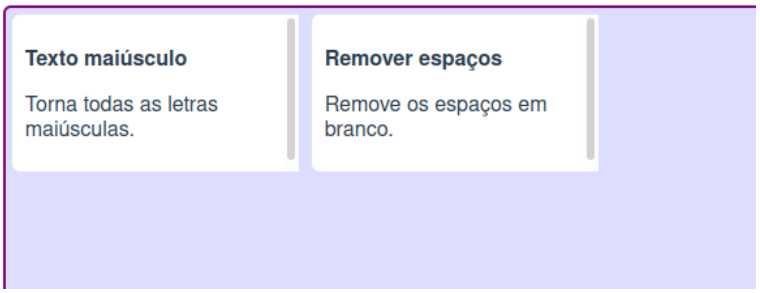

Figura 13: Área para manipulação de sequências já com duas ações definidas.

Este componente é a área de sequências onde as sequências são exibidas e manipuladas. Nesta região podemos adicionar uma ação, arrastando uma função e soltando-a, podemos trocar a ordem das ações e podemos removê-las, arrastando-as para a lixeira.

Figura 14: Lista dos módulos disponíveis para manipulação junto com suas funções.

Neste componente da **Figura 14**, temos a lista de módulos que foram criados através da API discutida anteriormente. Como podemos ver, cada módulo tem seu conjunto de funções que podem ser arrastadas para a área de sequências.

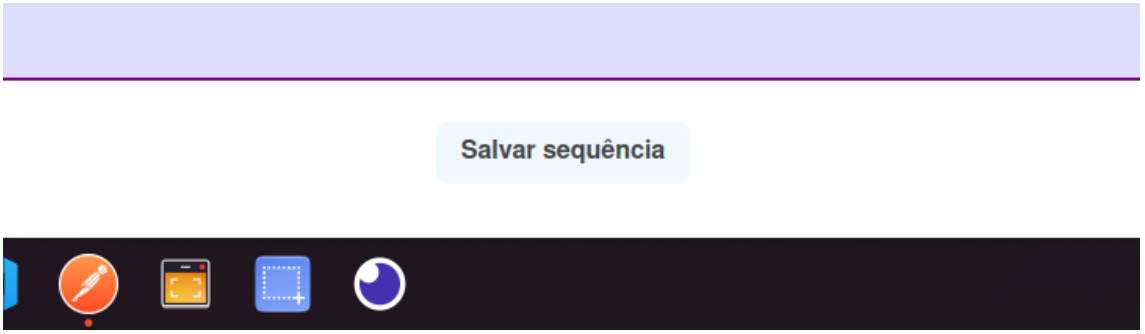

Figura 15: Botão para salvar a sequência definida.

Na **Figura 15** temos o botão para salvar a sequência formada. Ao ser clicado, o botão primeiro verifica se a sequência formada é válida. Aqui é importante lembrar que

temos os campos *argumentsType* e *responsesType*, e é através deles que vamos verificar se o dado que uma função entrega é do mesmo tipo que a função seguinte espera como entrada. Caso a sequência seja inválida, uma notificação de erro aparece na tela. Se a sequência for válida uma notificação informando que tudo deu certo é mostrada e a sequência é enviada para a API, sendo salva no banco de dados. Feito isso, assim que esta sequência for utilizada o comportamento definido será executado, não necessitando realizar uma atualização em produção.

Para a criação dos componentes das figuras 07, 08 e a lixeira, foi utilizada uma biblioteca de arrasta e solta chamada Sortable [8]. Com ela foi possível implementar as funcionalidades de arrastar de uma área e soltar em outra, mudar a ordem dos elementos e remover itens da área de sequências colocando-os na lixeira.

Sendo assim, tendo implementado a API de manipulação de comportamentos, a otimização em relação ao desempenho e a interface web, podemos analisar melhor o fluxo dos dados pela seguinte figura:

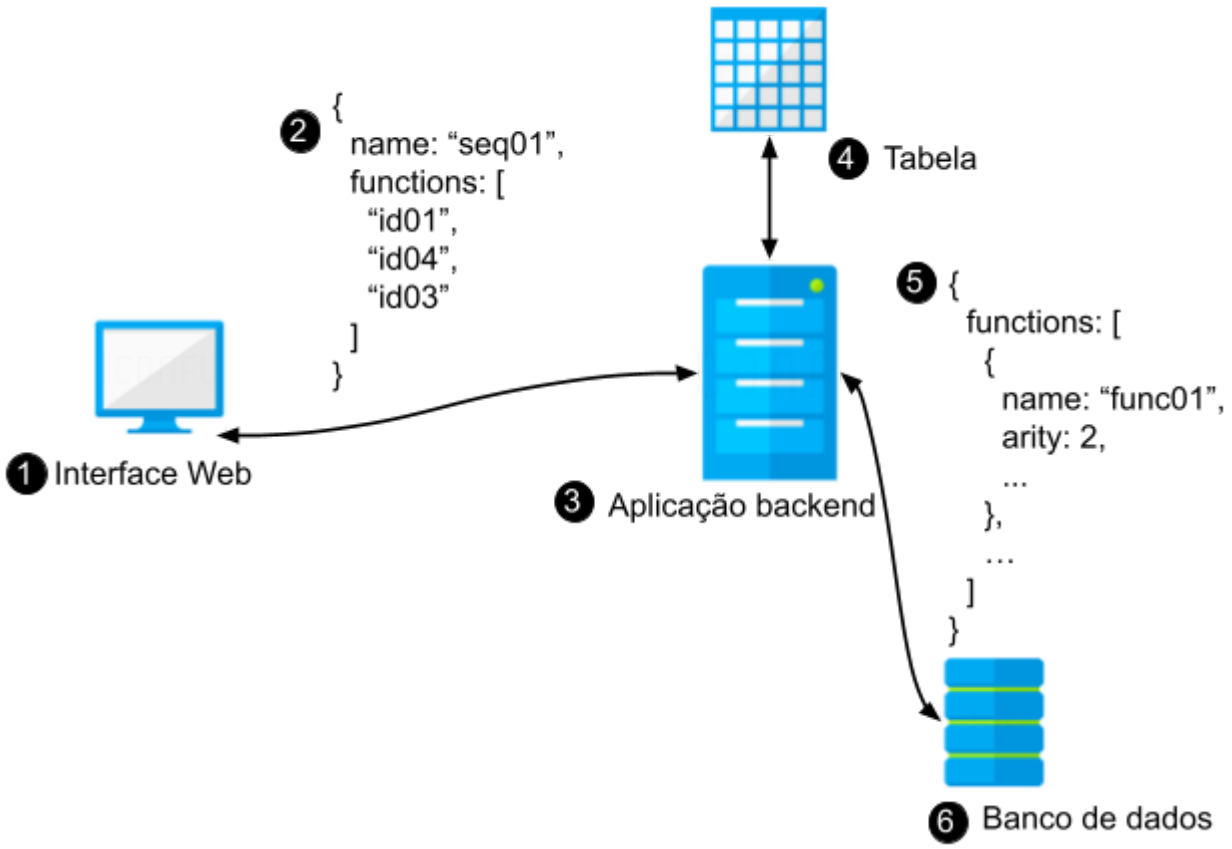

Figura 16: Fluxo na manipulação de sequências.

1. **Interface Web:** Assim que a interface web inicializar, requisições são feitas ao backend para buscar as sequências já existentes e assim serem mostradas na tela.

2. **Sequência:** Quando uma sequência é montada e precisa ser salva, ela é enviada ao backend, sendo informado seu nome e a sequência de identificadores das funções que compõem aquele comportamento.

3. **Aplicação backend:** Nesta aplicação a sequência é recebida e salva no banco de dados. Caso a sequência já exista na tabela de acesso rápido, ou seja já foi acessada em algum outro momento, a tabela também é atualizada com as modificações. A aplicação backend implementa tanto as funções para manipulação de comportamentos quanto as funções referentes aos produtos daquela empresa.

4. **Tabela:** Quando alguma sequência é acessada ela é salva na tabela, que fica na memória para acesso rápido, sendo assim ela é consultada antes de se precisar buscar as informações no banco de dados.

5. **Funções da sequência:** As funções que compõem a sequência são armazenadas, tanto na tabela quanto no banco de dados neste formato, para que a aplicação backend saiba quais funções deve acessar, pelo nome, e realizar as operações de forma correta a partir das informações fornecidas.

6. **Banco de dados:** Armazena as informações referentes as funções que são implementadas na aplicação backend e as sequências, quando se trata deste fluxo de criação e atualização de comportamentos.

Por fim, temos a arquitetura da implementação:

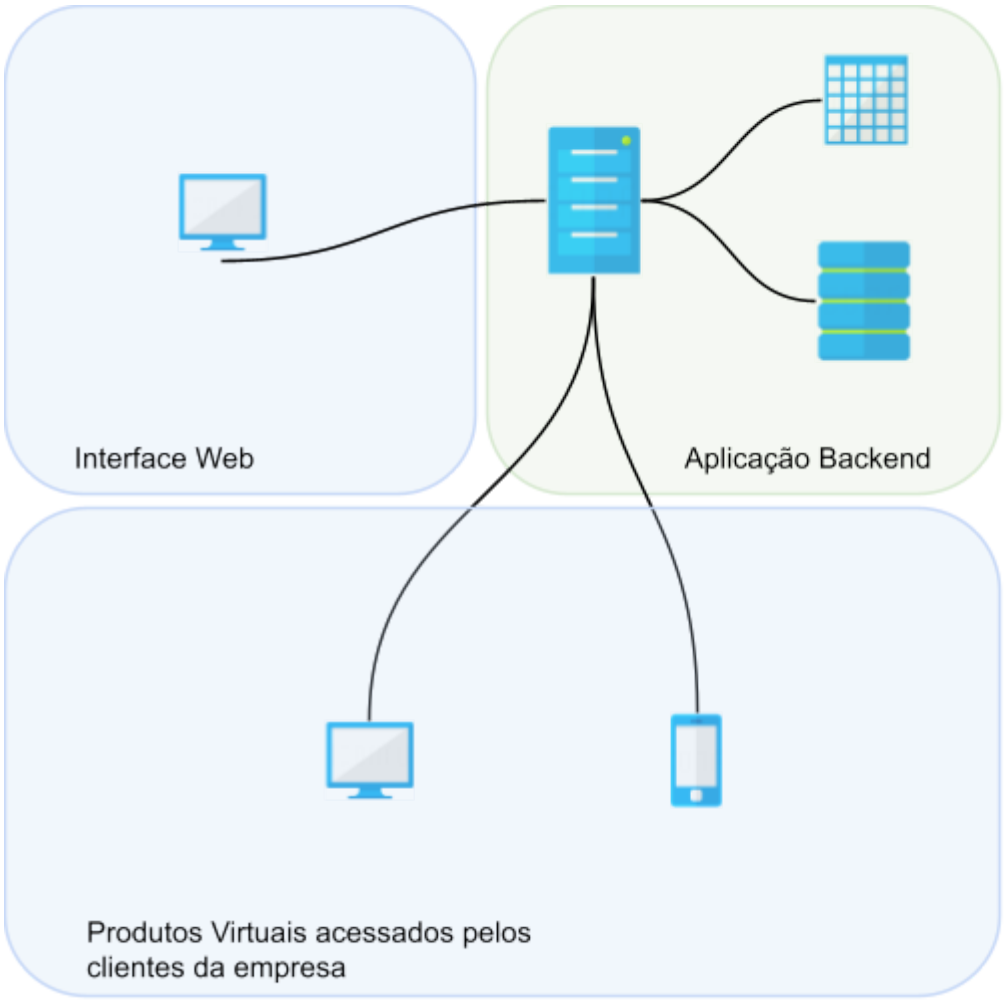

Figura 17: Arquitetura contendo a Interface Web, aplicação backend e os produtos virtuais.

5. Análise e Comparação

Neste capítulo será apresentado um caso de uso para que se entenda melhor como funciona a manipulação de comportamentos e em seguida uma comparação, frisando alguns pontos importantes, entre a ideia proposta neste trabalho e as plataformas já existentes apresentadas no capítulo 3 (Trabalhos relacionados).

5.1. Caso de uso

Para colocar os projetos anteriores em funcionamento algumas implementações foram adicionadas, portanto não são necessárias para o correto funcionamento da ideia proposta neste trabalho, servindo apenas como exemplos de aplicações reais.

Como primeiro exemplo, temos uma tela de registro que solicita informações básicas do usuário para realizar seu cadastro. Assim que o usuário clica no botão de se cadastrar, suas informações são enviadas ao backend para serem salvas em uma planilha. Dentre as informações que serão salvas está o atributo code (código do cliente), como mostrado na planilha da **Figura 18**.

1	Nome	Email	Código do cliente	Whatsapp
2	Fernando Torres	ftorres@gmail.com	FTORRES@GMAIL.COM	(12) 99233-2233
3	Bruna Aragão	brunaar@gmail.com	BRUNA ARAGÃO	(12) 99233-3200
4	Cecília Mendes	cmends@gmail.com	CECÍLIAMENDES	(12) 98320-0273
5	Igor Carneiro da Silva	spread.sheet@gmail.com	IGORCARNEIRODASILVA	(88) 88888-8888

Figura 18: Tabela com lista de dados de clientes cadastrados.

No primeiro registro vemos que o código do cliente é formado por seu email com todas as letras maiúsculas. Este comportamento pode ser formado pela seguinte sequência:

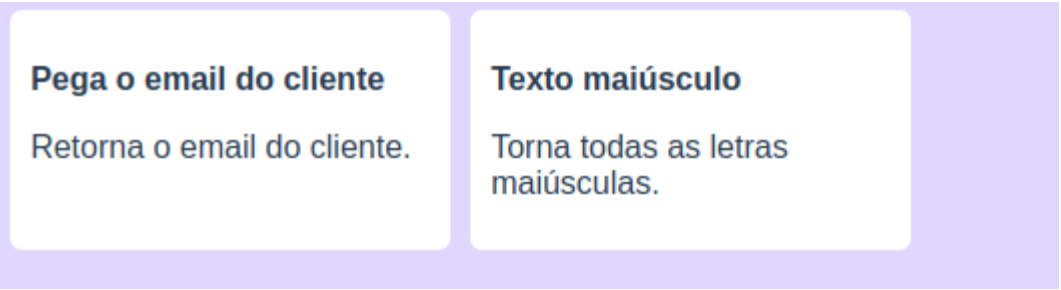

Figura 19: Sequência trabalhando com email do cliente.

Para que o email não seja colocado como código e sim o nome do cliente, apenas precisamos trocar o bloco do email do cliente para o bloco que retorna o nome do cliente, formando a sequência a seguir:

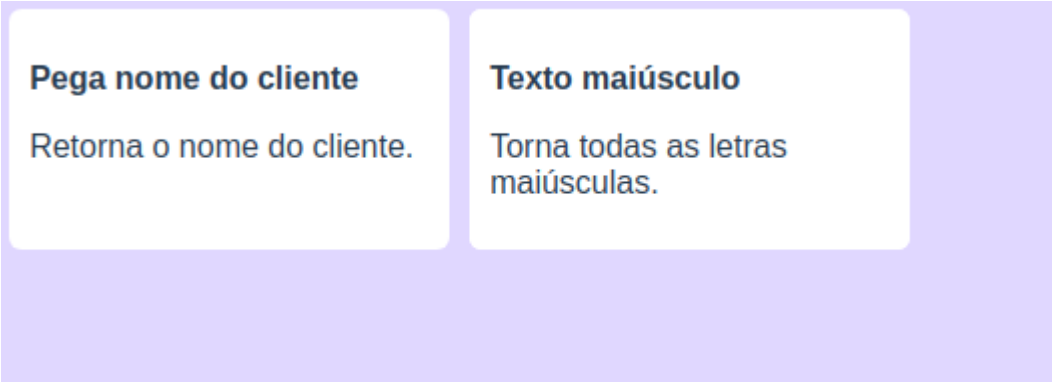

Figura 20: Sequência trabalhando com nome do cliente.

Supondo agora que ao utilizar o nome do cliente passamos a ter um problema por conter espaços em branco em seu código. Para corrigir isto, qualquer integrante pode adicionar o bloco que remove os espaços em branco de um texto, finalizando na sequência abaixo e sendo o comportamento dos registros 3 e 4 da planilha da **Figura 18**:

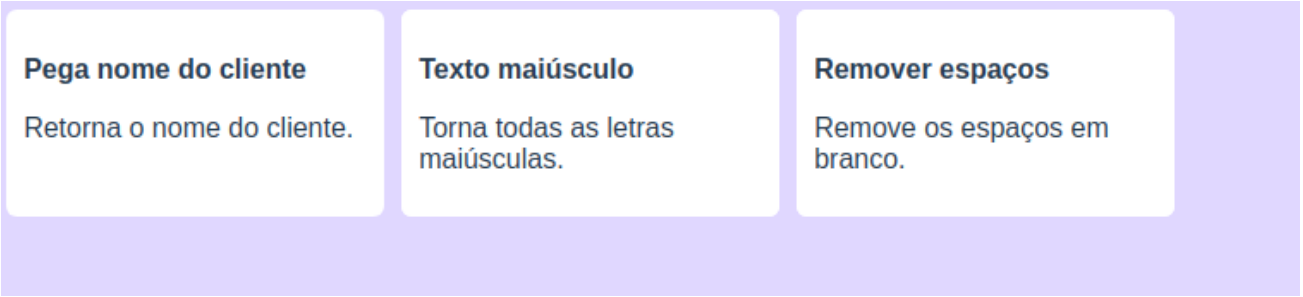

Figura 21: Sequência trabalhando com nome do cliente e agora com ação adicional de remoção de espaços em branco.

Um outro comportamento que podemos definir é como a aplicação web deve reagir ao registro dos clientes. Abaixo temos como exemplo a lista das ações implementadas:

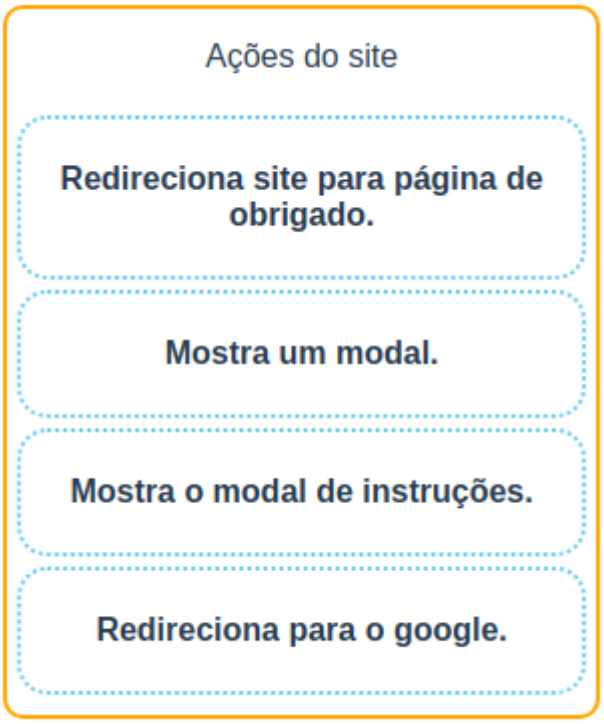

Figura 22: Lista de ações para serem utilizadas em uma aplicação web.

Ao finalizar o registro podemos redirecionar o cliente para uma tela de obrigado, mostrar algum modal ou até redirecionar para um site externo, como o google. Muitas outras ações, e mais complexas, podem ser implementadas, dependendo do que aquela aplicação deseja entregar.

5.2. Comparação com as plataformas existentes

A seguir temos uma comparação entre as plataformas já existentes, mencionadas no capítulo que trata do estado da arte, e a ideia proposta neste trabalho considerando alguns pontos importantes.

- **Requer conhecimento de tecnologia:** As plataformas já existentes são plataformas low-code ou no-code (plataformas que precisam de pouco ou nenhum código), mas que são desenvolvidas se esperando um conhecimento de lógica computacional e em alguns casos é recomendado ter até conhecimento básico de design. Já na ideia proposta neste trabalho a interface não necessitará de

conhecimento tecnológico algum, nem código, nem lógica computacional, nem design.

- **Estabelece um limite de linguagens:** As plataformas que já estão no mercado limita os desenvolvedores a uma lista de linguagens de programação que podem ser utilizadas para o desenvolvimento de suas aplicações. Na ideia apresentada neste trabalho os desenvolvedores poderão aplicar o que se propõe aos projetos já existentes independente da linguagem escolhida por eles, já que estaremos trabalhando com uma API manipulando um banco de dados. Isso faz com que se utilize a linguagem de programação mais apropriada aos problemas que a empresa pretende resolver.

- **Grátis e sem limitações:** A maioria das plataformas são pagas para serem utilizadas de forma comercial. As plataformas que são gratuitas possuem funcionalidades limitadas, tanto de acesso quanto de quantidade de contas que podem ser criadas. Já em relação ao que propõe este trabalho, como será algo desenvolvido pela própria empresa, não haverá custos nem a equipe ficará limitada em relação a acesso ao projeto e quantidade de contas criadas. O que precisa ter, caso a empresa ainda não esteja utilizando, é um domínio para hospedar a interface web.

- **Seguro tornar acessível a todos da equipe:** Para as plataformas existentes seria inseguro deixar toda a equipe acessá-la, dado que todo o projeto em questão e seus processos ficariam disponíveis para manipulação. Já na ideia aqui proposta temos um controle maior do que pode ser ou não manipulado por todos. Isso quem decide é a equipe. O que for escolhido para poder ser manipulado será apresentado na interface web, evitando maiores erros.

- **Atualização instantânea em produção:** Nestas plataformas já existentes as alterações realizadas, para que se esteja em produção, necessita que alguns outros procedimentos sejam executados. Já na ideia que este trabalho apresenta as alterações, assim que salvas, ficarão disponíveis em produção instantaneamente.

6. Conclusões e Trabalhos Futuros

Neste capítulo serão apresentadas quais as contribuições deste trabalho e quais melhorias são possíveis de serem feitas como trabalhos futuros.

6.1. Contribuições

Ao finalizar a implementação tanto da API de manipulação de comportamentos, quanto da interface web, conseguimos notar as seguintes contribuições:

- Agora os comportamentos dos produtos virtuais podem ser manipulados de forma simples, arrastando e soltando blocos sem que se tenha conhecimento de tecnologia.
- Qualquer integrante da equipe pode utilizar a interface.
- Certas atividades, antes destinadas apenas ao time de desenvolvimento, não ficarão para depois.
- Discussões em equipe mais ricas e resoluções mais assertivas, dado que agora a bagagem de conhecimento de cada integrante aumentou ao adquirir mais conhecimento a respeito de quais ações seus produtos podem ou não realizar.
- Atualizações em produção mais rápidas.

6.2. Trabalhos Futuros

A estrutura de sequências aqui proposta não está no seu modelo ideal. Mesmo realizando a otimização discutida no capítulo 5 (Análise e Comparação), ainda há melhorias a se fazer, mostradas a seguir:

6.2.1. Sequência de sequências

A estrutura tanto da interface web, no que se refere a montagem das sequências, quanto da API não possibilita a formação de sequências contendo outras sequências.

Lembrando que sequências (comportamentos) são listas de funções (ações). Como exemplo, se já tivermos criado a sequência que transforma todos os caracteres de um texto em maiúsculo e após isso remove todos os espaços em branco, caso queiramos realizar este mesmo comportamento dentro de uma outra sequência devemos arrastar os dois blocos que constituem esta sequência já criada, quando poderíamos apenas arrastar e soltar a mesma. A forma descrita neste trabalho apenas possibilita arrastar e soltar funções, não sequências.

6.2.2. Implementar API de sequências em sua própria aplicação

Um problema a se notar, para empresas que possuem mais de uma aplicação backend (que geralmente é o que acontece), é que se quisermos utilizar a ideia de manipulação de comportamentos em todas estas suas aplicações, devemos implementar a API de sequências em cada uma delas, o que dificultaria a manutenção e a integração da interface web com cada uma destas aplicações. A ideia então seria criar uma aplicação responsável apenas pelas funcionalidades referentes à manipulação de sequências, e esta estaria interligada com as demais aplicações. Assim, a manutenção seria realizada em apenas um projeto e a integração com a interface web seria simplificada, deixando de se usar vários endpoints diferentes.

BIBLIOGRAFIA

[1] DAVE THOMAS, “Programming Elixir ≥ 1.6_ Functional __Concurrent __Pragmatic __Fun”, capítulos 7 Lists and Recursion, 12 Control Flow e 16 Nodes - The Key to Distributing Services.

[2] CHRIS McCORD, BRUCE TATE, JOSÉ VALIM, “Programming-Phoenix - 1.4”, capítulos 2 The Lay of the Land, 3 Controllers e 4 Ecto and Changesets.

[3] ELIXIR, a dynamic, functional language for building scalable and maintainable applications. Disponível em: https://elixir-lang.org/. Acesso em: 16 de junho de 2021.

[4] ERLANG, Build massively scalable soft real-time systems. Disponível em: https://erlang.org/doc/search/.

[5] PHOENIX, Peace of mind from prototype to production. Disponível em: https://phoenixframework.org/.

[6] ECTO SCHEMA, Defines a schema. Disponível em: https://hexdocs.pm/ecto/Ecto.Schema.html

[7] VUE JS, The Progressive JavaScript Framework. Disponível em: https://vuejs.org/v2/guide/. Acesso em: 17 de julho de 2021.

[8] SORTABLE JS, JavaScript library for reorderable drag-and-drop lists. Disponível em: http://sortablejs.github.io/Sortable. Acesso em 17 de julho de 2021.

[9] These Modern Programming Languages Will Make You Suffer - Welcome to the ultimate rating of modern programming languages. Publicado em: 7 de dezembro de 2020. Disponível em: https://betterprogramming.pub/modern-languages-suck-ad21cbc8a57c. Acesso em: 18 de junho de 2021.

[10] OUTSYSTEMS, Build Applications Fast, Right and For the Future. Disponível em: https://www.outsystems.com/pt-br/platform/. Acesso em: 06 de julho de 2021.

[11] KODULAR, Much more than a modern app creator without coding. Disponível em: https://www.kodular.io/. Acesso em: 06 de julho de 2021.

[12] THUNKABLE. Fast apps, no coding. Disponível em: https://thunkable.com. Acesso em: 06 de julho de 2021.

[13] BUBBLE. The best way to build web apps without code. 2021. Disponível em: https://bubble.io. Acesso em: 06 de julho de 2021.

[14] Erlang Term Storage (ETS). Disponível em: https://elixirschool.com/pt/lessons/specifics/ets/. Acesso em: 08 de julho de 2021.

APÊNDICES

Apêndice A - Pseudo-código e código em Elixir do cálculo de aridade de uma sequência

```
1.  início “cálculo de aridade de uma sequência”
2.
3.      var aridade_atual <- 0;
4.      var tipo_da_resposta <- [ ];
5.
6.      enquanto função <- sequência faça
7.          se tipo_da_resposta == [ ] então
8.              aridade_atual <- aridade_atual + função.aridade;
9.          senão então
10.             aridade_atual <- aridade_atual + função.aridade - 1;
11.         fim senão
12.
13.         tipo_da_resposta <- função.tipo_da_resposta;
14.     fim enquanto
15. fim
```

Detalhamento do pseudocódigo:

- **linha 3**: declarando variável que armazenará o valor da aridade corrente e inicializando-a com o valor 0;
- **linha 4**: declarando variável que armazenará o valor do tipo de resposta da função anterior. O tipo de resposta [ ] (lista vazia) indica que a função não retorna resposta;
- **linha 6**: início do ciclo em que a cada iteração um item da sequência será armazenado em ‘função’;
- **linha 7**: verificando se a função da iteração anterior não retorna resposta;
- **linha 8**: incrementando o valor da aridade corrente com o valor da aridade da função daquela iteração;
- **linha 9**: caso a função anterior retorne alguma resposta;

- **linha 10**: incrementando o valor da aridade corrente com o valor da aridade da função daquela iteração menos 1, já que como a função anterior retorna resposta ela será utilizada como primeiro argumento da função corrente;
- **linha 13**: armazenando o tipo de resposta da função corrente para que seja analisado na próxima iteração. Caso seja a última iteração não importa, para o cálculo da aridade, se a função retorna ou não alguma resposta.

Função do cálculo de aridade em Elixir:

```
1.  def count_arity_sequence(sequence) do
2.
3.    total_arity =
4.      Enum.reduce(sequence, {0, @type_response_no_reply} , fn block, {current_arity,
    type_response} ->
5.        cond do
6.          block.arity == 0 ->
7.            {current_arity, block.responsesType}
8.
9.          type_response == @type_response_no_reply ->
10.           {current_arity + block.arity, block.responsesType}
11.
12.         true ->
13.           {current_arity + block.arity - 1,     block.responsesType}
14.       end
15.     end)
16.
17.   {result_arity, _type_reponse} = total_arity
18.
19.   result_arity
20. end
```

Apêndice B - Pseudo-código e código em Elixir da execução de uma sequência

1. **início** “execução de uma sequência”
2.
3. **var** resultado <- null;
4. **var** tipo_da_resposta <- [ ];
5. **var** parâmetros <- parâmetros_fornecidos;
6. **var** parâmetros_da_função <- [ ]
7.
8. **enquanto** função <- sequência **faça**
9. **se** função.aridade == 0 **então**
10. resultado <- executar(função);
11. **senão se** tipo_da_resposta == [ ] **então**
12. parâmetros_da_função <- **retorna** lista({função.aridade} primeiros valores de parâmetros);
13. parâmetros <- **retorna** parâmetros sem os {função.aridade} primeiros valores;
14. resultado <- executar(função, parâmetros_da_função);
15. **senão então**
16. parâmetros_da_função <- **retorna** lista(resultado e {função.aridade - 1} primeiros valores de parâmetros);
17. parâmetros <- **retorna** parâmetros sem os {função.aridade - 1} primeiros valores;
18. **fim senão**
19.
20. tipo_da_resposta <- função.tipo_da_resposta;
21. **fim enquanto**
22. **retornar** resultado
23. **fim**

Detalhamento do pseudocódigo:

- **linha 3**: declarando variável que armazenará o valor do resultado da função corrente da iteração e inicializando-a com o valor nulo (null);

- **linha 4**: declarando variável que armazenará o valor do tipo de resposta da função anterior. O tipo de resposta [ ] (lista vazia) indica que a função não retorna resposta;
- **linha 5**: declarando variável que armazenará a lista de parâmetros para a execução daquela sequência. Por exemplo, uma sequência pode receber a lista de parâmetros [23, “mensagem padrão”, {nome: “Flávio”, idade: 17}];
- **linha 6**: declarando variável que armazenará a lista de parâmetros necessários para a execução da função corrente;
- linha 8: início do ciclo em que a cada iteração um item da sequência será armazenado em ‘função’;
- **linha 9**: verifica se a aridade da função corrente é 0;
- **linha 10**: a função da iteração é executada e o resultado armazenado na variável resultado;
- **linha 11**: caso a aridade da função não seja 0 verificar se o tipo de resposta da função da iteração anterior é [ ], ou seja não retorna resposta;
- **linha 12**: se a aridade da função é diferente de 0 significa que ela precisa de argumentos para ser executada, e estes argumentos serão retirados da variável parâmetros. A lista com os argumentos para a execução da função deverá ser do tamanho de sua aridade;
- **linha 13**: como os primeiros valores da variável parâmetros foram utilizados na execução da função eles devem ser removidos e assim a variável parâmetros atualizada;
- **linha 14**: executar função, agora passando os parâmetros necessários, e salvar a sua resposta em resultado;
- **linha 15**: caso nenhuma condição anterior seja satisfeita, significa que a função corrente precisa de parâmetros para ser executada (aridade diferente de 0) e a função anterior retornou alguma resposta;
- **linha 16**: forma lista de argumentos para a execução da função, agora colocando a resposta da função anterior como primeiro valor desta lista;
- **linha 17**: novamente, remover da variável parâmetros os valores que foram passados para a execução da função;
- **linha 20**: armazenando o tipo de resposta da função corrente para que seja analisado na próxima iteração;
- **linha 22**: retornar o valor final computado pela sequência.

Função da execução de uma sequência em Elixir:

```
1.  def process_sequence([], _params, _response_type, result) do
2.    {:ok, result}
3.  end
4.
5.  def process_sequence(sequence, params, response_type, result) do
6.    # Pegando o primeiro bloco da lista.
7.    [function | tail] = sequence
8.
9.  # Transformando nomes de módulo e função em atoms.
10. module_atom = String.to_atom("Elixir." <> function.module.module)
11. function_atom = String.to_atom(function.function)
12.
13. cond do
14.   function.arity == 0 ->
15.     block_operation_response = apply(module_atom, function_atom, [])
16.
17.     process_sequence(tail, params, function.responsesType, block_operation_response)
18.
19.   response_type == @type_response_no_reply ->
20.     block_operation_response = apply(module_atom, function_atom, Enum.slice(params,
    0..(function.arity - 1)))
21.
22.     process_sequence(tail, Enum.drop(params, function.arity), function.responsesType,
    block_operation_response)
23.
24.   true ->
25.     block_operation_response = apply(module_atom, function_atom, Enum.slice([result |
    params], 0..(function.arity - 1)))
26.
27.     process_sequence(tail, Enum.drop(params, function.arity - 1), function.responsesType,
    block_operation_response)
28.  end
29. end
```