Mira-Editor

Ferramenta de prototipação auxiliar ao Mira Framework

INF 2102 - Projeto Final de Programação

João Victor Magela

1421957

Sumário

1 - Objetivo

O Mira-Editor é um framework para geração de código para o Mira[1]. Seu objetivo é facilitar o desenvolvimento de aplicações para o Mira através de prototipação. A sintaxe de criação das aplicações no Mira é no formato *JSON*[2], com isso usuários com pouco conhecimento técnico tendem a ter uma maior dificuldade de utilização. O Mira-Editor visa gerar o código para o Mira através do sistema de *Mockup*, onde os usuários podem usar elementos visuais para simbolizar o que realmente desejam.

2 – Arquitetura

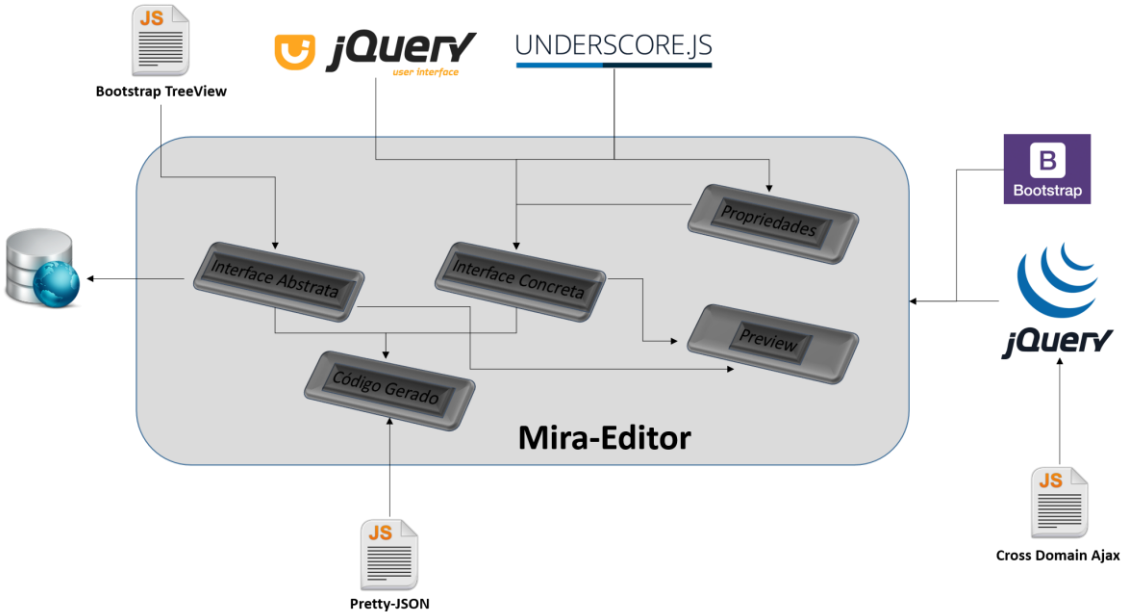

Figura 1 - Arquitetura do Mira-Editor

O Mira-Editor é composto por camadas que conversam entre si. Essa conversa tem dois objetivos principais: Gerar o código da aplicação para o Mira e exibir uma prévia do resultado. Com isso, o usuário pode ter mais facilidade em construir aplicações e mais convicção de que está fazendo de forma correta. Como podemos ver a Figura 1, o Mira-Editor é formado pelas camadas Interface Abstrata, Interface Concreta, Propriedades, Código Gerado e *Preview*. A Interface Abstrata é responsável por determinar a hierarquia e dados dos elementos da aplicação, sendo que dados podem ser oriundos de uma base dados externa. Já a Interface Concreta mapeia essa hierarquia em elementos concretos. Com as informações das duas Interfaces é possível gerar o código para o Mira. A camada Propriedades é responsável por alterar as propriedades de um elemento concreto. Já a camada *Preview* coleta as informações da Interface Concreta é renderiza o HTML, utilizando também dados inseridos nos elementos da Interface Abstrata.

2.1 – Bibliotecas, Plug-ins e Frameworks utilizados

Este capítulo visa contextualizar as bibliotecas e plug-ins que serão utilizados na implementação do projeto. O objeto é mostrar as dependências, tanto da aplicação de um modo geral, quanto de camadas específicas.

2.1.1 – jQuery[3]

É uma biblioteca utilizada para manipulação do *DOM[4]*. Seu objetivo é abstrair ao máximo operações relacionadas do *DOM* através de métodos. Além disso, há a facilidade em trabalhar com requisições *AJAX*. Um problema com relação à requisição *AJAX* é que os navegadores não aceitam por padrão o *Cross Domain* (requisições externas ao atual domínio), com isso, é necessária a reimplementação do método *$.ajax()*, que é feito pelo plug-in *Cross Domain Ajax[5]*.

2.1.2 – jQuery UI[6]

É uma extensão do *jQuery* e tem como objetivo facilitar a interação do usuário com a aplicação, através de um conjunto de efeitos. Sua aplicação será ligada a camada Interface Concreta, onde será possível arrastar e redimensionar os elementos.

2.1.3 – Underscore.js[7]

É uma biblioteca com inúmeros funcionalidades úteis para o desenvolvimento de aplicações. Sua principal utilização no Mira-Editor é na criação de templates, que serão utilizados nas camadas Interface Abstrata e Propriedades. Ou seja, para cada elemento concreto há um *template* referente à sua renderização e um *template* referente ao seu formulário de propriedades.

2.1.4 – Twitter Bootstrap[8]

É um framework de desenvolvimento *front-end* que facilita a estilização de elementos do HTML através de classes, além de fornecer componentes e plug-ins. A camada visual do Mira-Editor utiliza em sua grande parte os *Twitter Bootstrap*.

2.1.5 – BootstrapTreeView[9]

Com base no Twitter Bootstrap, seu objetivo é exibir elementos de forma hierárquica. Essa plug-in será útil para exibição da Interface Abstrata.

2.1.6 – Pretty-JSON[10]

É uma reimplementação do método *JSON.stringify()*, pois este retorna um código muito extenso. O objetivo do Pretty-JSON é deixar o código gerado ainda mais compacto.

3 - Especificação

Para detalhas os requisitos do Mira-Editor, optou-se por utilizar diagramas de caso de uso. Os casos de uso serão separados conformes sua camada. Para cada caso de uso será mostrado o detalhamento do mesmo.

3.1 – Projeto

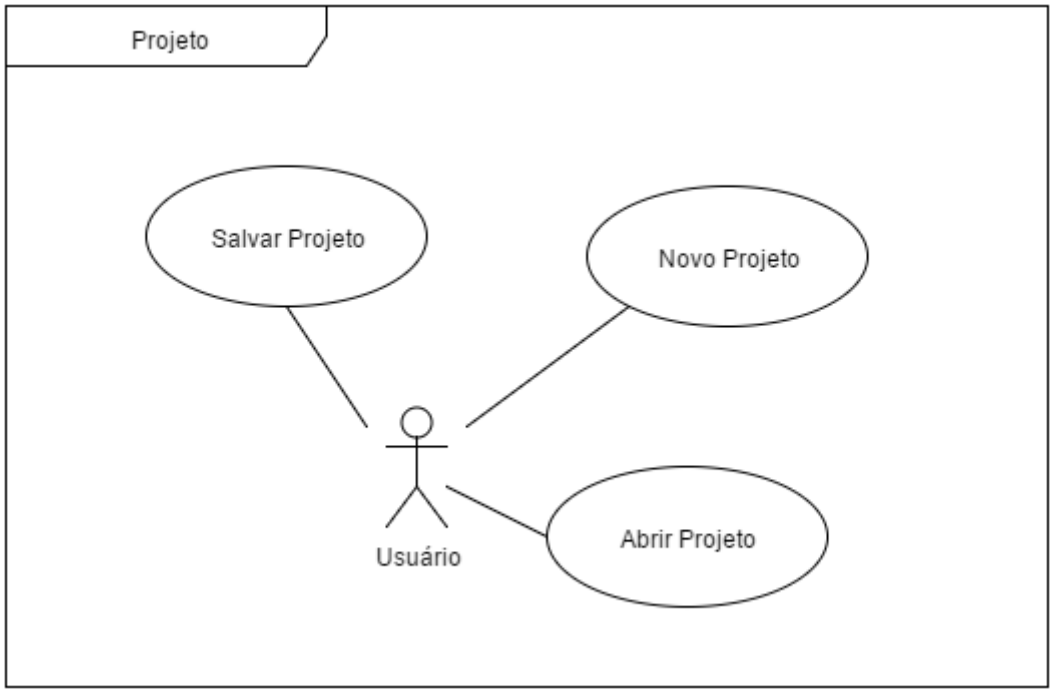

Figura 2 - Caso de uso Projeto.

Casos de uso responsável por gerir o projeto, tendo as funcionalidades “Salvar”, “Novo” e “Abrir”.

3.1.1 - Salvar Projeto

DESCRIÇÃO SUCINTA

O sistema salva todas as informações atuais do projeto, as insere no arquivo Interface.JSON e faz o download para o usuário.

ATORES

1. Usuário

PRÉ-CONDIÇÕES

Não há.

FLUXO BÁSICO
1. O Usuário clica no botão “Salvar”.
2. O sistema faz o download do arquivo Interface.JSON.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
Não há.

3.1.2 – Novo Projeto

DESCRIÇÃO SUCINTA
O sistema recarrega a projeto com os valores padrões.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
1. O Usuário clica no botão “Novo”.
2. O sistema exibe a mensagem para confirmação da operação.
   2.1. Caso não seja confirmado, o caso de uso é encerrado.
   2.2. Caso seja confirmado, é carregado um projeto com valores padrões
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
Não há.

3.1.3 – Abrir Projeto

DESCRIÇÃO SUCINTA
O sistema cria um projeto com os valores contidos no arquivo passado pelo usuário.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO

1. O Usuário clica no botão “Abrir”.
2. O usuário seleciona o arquivo que deseja abrir.
   2.1. Caso a **RN1** não seja atendida, é informada a mensagem “O arquivo deve ser no formado JSON”.
   2.2. Caso a **RN2** não seja atendida, é informada a mensagem “O arquivo não é válido”.
   2.3. O sistema cria o projeto com as informações contidas no arquivo.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS

Não há.

REGRAS DE NEGÓCIO

(**RN1**) O arquivo escolhido deve conter a extensão “.json”.

(**RN2**) O arquivo deve estar com a estrutura correta.

3.2 – Interface Abstrata

Camada responsável por exibir a hierarquia dos elementos e seus respectivos dados.

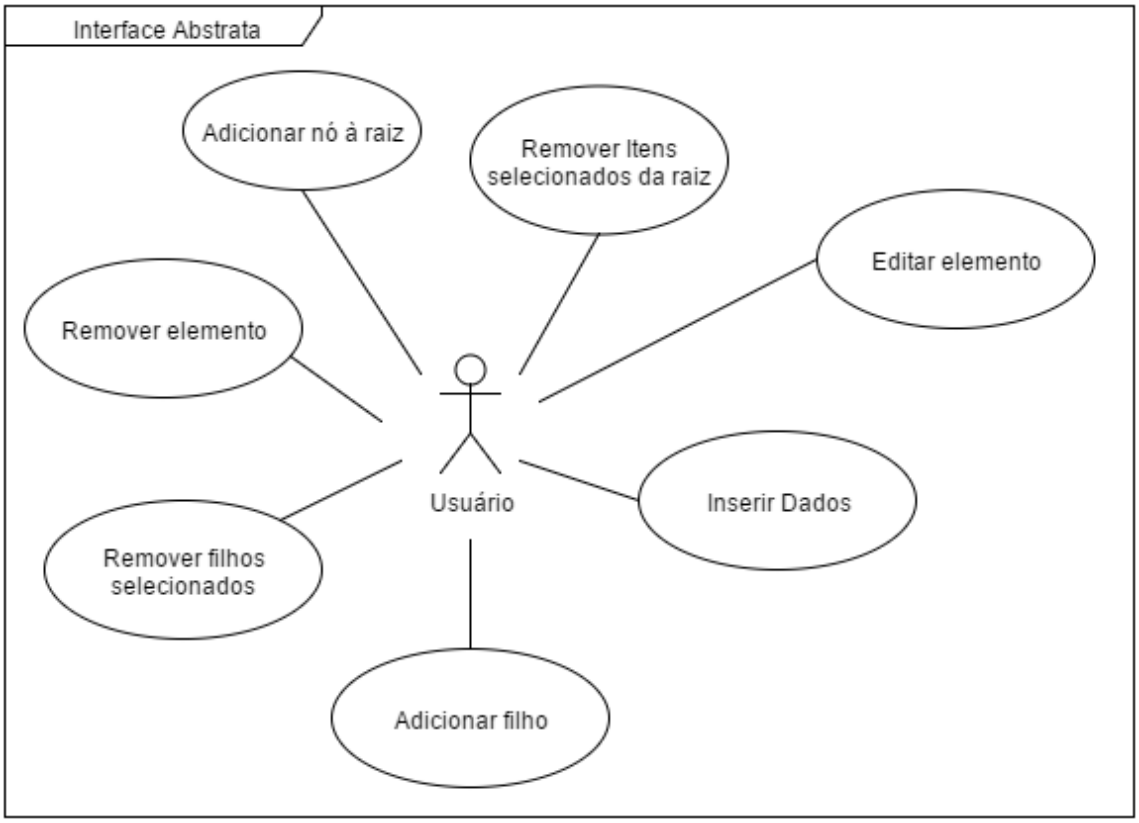

Figura 3 - Caso de uso Interface Abstrata.

3.2.1 – Adicionar nó à raiz

DESCRIÇÃO SUCINTA

Adiciona um item no topo da hierarquia.

ATORES

1. Usuário

PRÉ-CONDIÇÕES

Não há.

FLUXO BÁSICO

1. O Usuário clica no botão de adicionar nó à raiz.
2. O sistema gera um item no modo editável.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS

(A1) Alternativa ao passo 2 – Usuário pressiona a tecla “Enter” ou “Tab”

1.1. O sistema confirma a operação.
1.2. O item deixa de ser editável.

(A1) Alternativa ao passo 2 – Usuário pressiona a tecla “Esc”
2.1. O sistema exclui o item.

REGRAS DE NEGÓCIO
(**RN1**) O arquivo escolhido deve conter a extensão “.json”.

(**RN2**) O arquivo deve estar com a estrutura correta.

3.2.2 – Remover itens selecionados da raiz

DESCRIÇÃO SUCINTA
Remove todos os itens que estão selecionados e esteja no topo da hierarquia

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
4. O Usuário seleciona os itens que deseja excluir marcado o *checkbox*.
5. O usuário clica no botão remoção.
6. O sistema exclui os itens da árvore.
7. O caso de uso é encerrado

FLUXOS ALTERNATIVOS
(A1) Alternativa ao passo 1 – Não há itens na árvore.
1.3. O caso de uso é encerrado.

REGRAS DE NEGÓCIO
Não há.

3.2.3 – Editar Elemento

DESCRIÇÃO SUCINTA
Edita um item da árvore

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve seguir **RN1**.

FLUXO BÁSICO
1. O Usuário clica de edição do item da árvore.
2. O sistema passa o estado desse item para editável.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
(A1) Alternativa ao passo 2 – Usuário pressiona a tecla "Enter" ou "Tab"
2.1. O sistema confirma a edição.
2.2. O item deixa de ser editável.

(A1) Alternativa ao passo 2 – Usuário pressiona a tecla "Esc"
2.1. O sistema cancela a operação de edição.

REGRAS DE NEGÓCIO
(**RN1**) A árvore deve conter pelo menos um item.

3.2.4 – Adicionar filho

DESCRIÇÃO SUCINTA
Adiciona um item como filho de outro na hierarquia.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve seguir **RN1**.

FLUXO BÁSICO
1. O Usuário clica de edição de adicionar um filho do item desejado.
2. O sistema cria um item como filho com o estado editável.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
(A1) Alternativa ao passo 2 – Usuário pressiona a tecla "Enter" ou "Tab"
2.1. O sistema confirma a operação.
2.2. O item deixa de ser editável.

(A1) Alternativa ao passo 2 – Usuário pressiona a tecla “Esc”
2.1. O sistema exclui o item.

REGRAS DE NEGÓCIO
(**RN1**) A árvore deve conter pelo menos um item.

3.2.5 – Inserir Dados

DESCRIÇÃO SUCINTA
Possibilita ao usuário atribuir informações um determinado item.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve seguir **RN1**.

FLUXO BÁSICO
1. O Usuário clica de edição obter informações do item desejado.
2. O sistema abre um modal com um formulário.
3. O usuário insere o link ou expressão e clica no botão para processar o *Datasource*.
   3.1. Se a **RN2** for atendida:
      3.1.1. O sistema insere os atributos disponíveis no elemento.
      3.1.2. Exibe o campo parse.
      3.1.3. Exibe os atributos disponíveis na tabela.
   3.2. O usuário insere o *Parse* e clica no botão para processá-lo.
   3.3. Se **RN3** for atendida:
      3.3.1. O sistema insere os atributos disponíveis no elemento.
      3.3.2. Exibe os atributos disponíveis na tabela.
   3.4. Usuário clica no botão de confirmação.
4. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
(A1) Alternativa ao passo 3.1 – RN2 não foi atendida.
3.1.1. Exibe a mensagem de erro alertando que a expressão ou link está incorreto.
3.1.2. Oculta o campo parse.

(A2) Alternativa ao passo 3.3 – RN3 não foi atendida.
3.3.1. O sistema informa que a expressão não é válida.

(A3) Alternativa ao passo 3.4 – Usuário fecha o modal sem confirmar.
3.4.1. O sistema cancela a operação
3.4.2. O caso de uso é encerrado

REGRAS DE NEGÓCIO
(**RN1**) A árvore deve conter pelo menos um item.
(**RN2**) O link ou expressão deve ser válido
(**RN3**) A expressão deve ser válida.

3.2.6 – Remover Elemento

DESCRIÇÃO SUCINTA
Remove um item e seus filhos.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve seguir **RN1**.

FLUXO BÁSICO
1. O Usuário clica no botão de remoção do item desejado.
2. O sistema exclui o item e seus filhos.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
(**RN1**) A árvore deve conter pelo menos um item.

3.2.7 – Remover filhos selecionados

DESCRIÇÃO SUCINTA
Remove os filhos de um determinar item.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve seguir **RN1**.

FLUXO BÁSICO
1. O Usuário seleciona os filhos que deseja excluir.
2. O usuário clica no botão de remover filhos do pai.
3. O sistema exclui os filhos selecionados.
4. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
(**RN1**) A árvore deve conter pelo menos um item.

3.3 – Interface Concreta

Representa o protótipo da aplicação, onde os usuários poderão trabalhar com os elementos concretos.

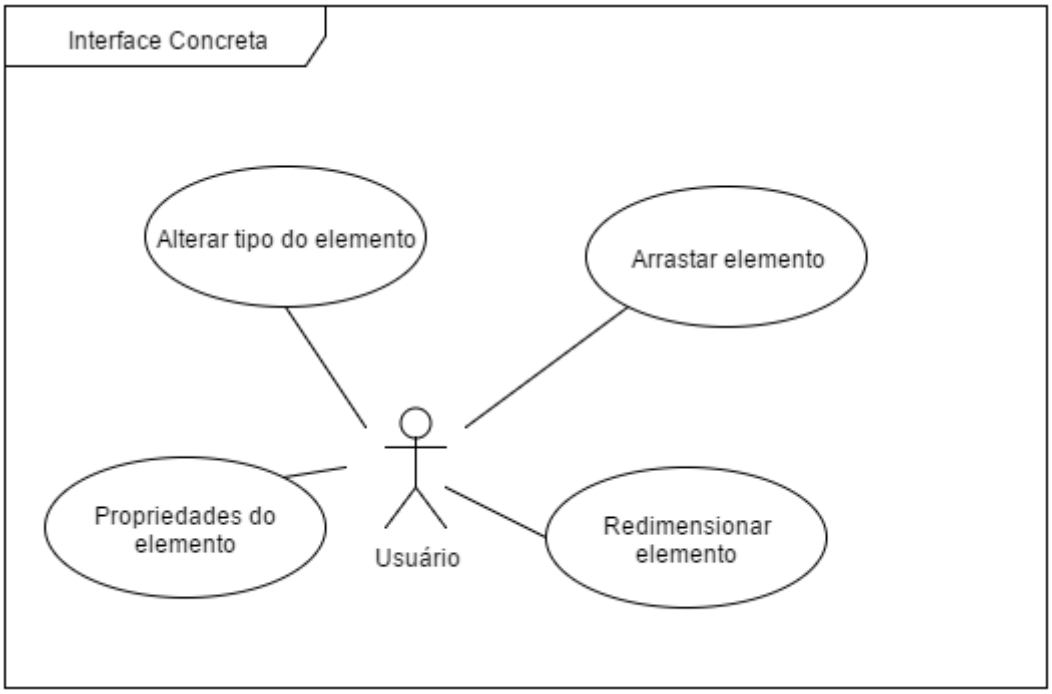

Figura 4 - Caso de uso Interface Concreta.

3.3.1 – Alterar tipo do elemento

DESCRIÇÃO SUCINTA
Permite ao usuário alterar o tipo do elemento, selecionado assim o tipo que melhor representa o que realmente deseja expressar.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
1. O Usuário seleciona qual o tipo que deseja.
2. O sistema detecta o template para esse item e renderiza no protótipo.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há

REGRAS DE NEGÓCIO
Não há.

3.3.2 – Propriedades do elemento

DESCRIÇÃO SUCINTA
Acessa o formulário de propriedades do elemento.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há

FLUXO BÁSICO
1. O Usuário clica no botão de propriedades do item desejado.
2. O sistema abre o formulário para edição das propriedades, segundo a **RN1**.
3. Executa o caso de uso **Salvar Propriedades**.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
(RN1) Se o elemento abstrato tiver algum "*Bind*", no campo valor deve aparecer **"$bind"**.

3.3.3 – Arrastar Elemento

DESCRIÇÃO SUCINTA
Arrasta um determinado elemento internamente ao seu pai.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
1. O pressiona o botão para arrastar o elemento e o arrasta.
2. O arraste deve seguir a regra **RN1**.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS

REGRAS DE NEGÓCIO
(**RN1**) O elemento só pode ser arrastado dentro do seu pai. Caso o elemento não possua um pai, o próprio protótipo será o seu pai.

3.3.4 – Redimensionar elemento

DESCRIÇÃO SUCINTA
Altera as dimensões de um determinado elemento.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
1. O Usuário pressiona as bordas do elemento e arrasta para poder redimensionar.
2. O redimensionamento deve seguir a **RN1** e **RN2**.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
(**RN1**) A altura mínima deve ser maior que a soma da altura dos filhos do elemento.
(**RN2**) A largura mínima deve ser 200px.

3.4 - Propriedades

Permite ao usuário alterar as propriedades de um determinado elemento.

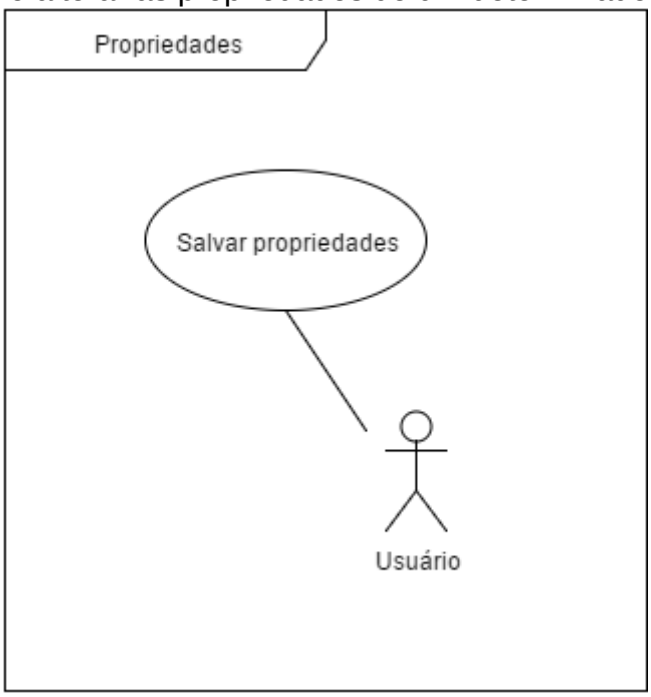

Figura 5 - Caso de uso Propriedades.

3.4.1 – Salvar propriedades

DESCRIÇÃO SUCINTA
Altera as propriedades de um elemento concreto.

ATORES
1. Usuário

PRÉ-CONDIÇÕES
Deve ter executado o caso de uso **Propriedades do elemento**.

FLUXO BÁSICO
1. O Usuário preenche o formulário e clica em salvar.
2. O sistema salvar as alterações.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
(A1) Alternativa ao passo 1 – Usuário clica no botão cancelar
1.1. O sistema mantem os valores das propriedades do elemento.

REGRAS DE NEGÓCIO
Não há.

3.5 – Código Gerado

Permite ao usuário visualizar o código gerado para a Interface Abstrata e Concreta.

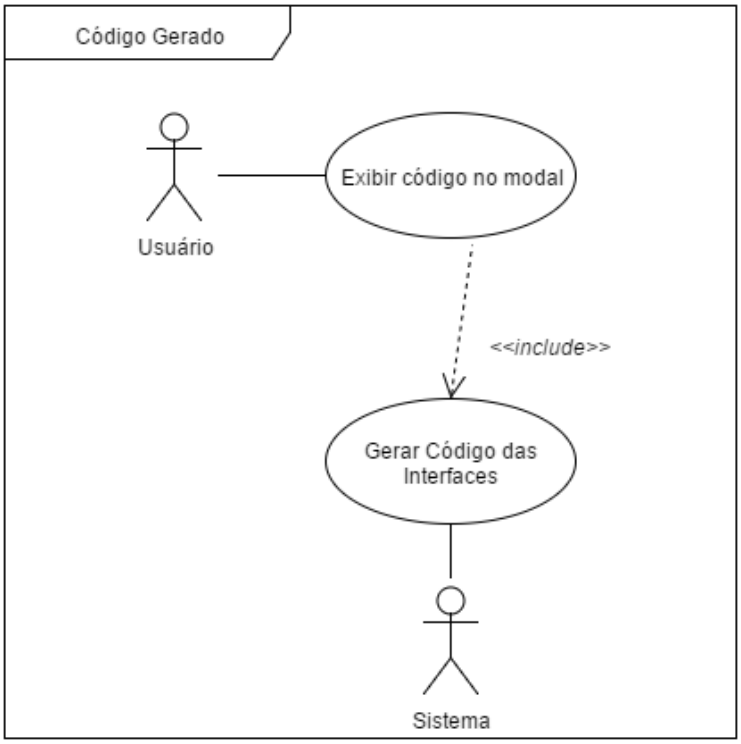

Figura 6 - Caso de uso Código Gerado.

3.5.1 – Gerar código das Interfaces

DESCRIÇÃO SUCINTA
Gera o código da Interface Abstrata e Concreta e as exibe para o Usuário.

ATORES
1. Sistema.

PRÉ-CONDIÇÕES
Não há.

FLUXO BÁSICO
1. O sistema coleta as informações da Interface Abstrata e Concreta.
2. Gera as informações de acordo com as informações coletadas.
3. Exibe para o usuário.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
Não há.

3.5.2 – Exibir código no *modal*

DESCRIÇÃO SUCINTA
Permite que o usuário veja o código em uma área maior.

ATORES
1. Usuário.

PRÉ-CONDIÇÕES
Ter executado o caso de uso “**Gerar código das Interfaces”** pelo menos uma vez.

FLUXO BÁSICO
1. O usuário clica no botão para visualizar código desejado.
2. Abre-se o modal com o código gerado.
3. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
Não há.

3.6 – *Preview*

Exibe uma prévia do trabalho feito pelo usuário, afim de validar seu trabalho.

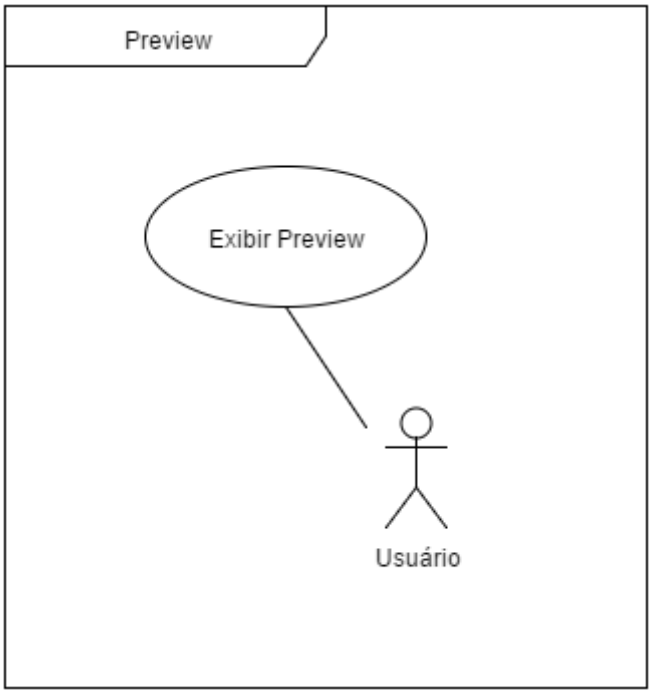

Figura 7 - Caso de uso Preview.

3.6.1 – Exibir *Preview*

DESCRIÇÃO SUCINTA
Permite que o usuário veja como o código gera irá se comportar numa aplicação Mira.

ATORES
1. Usuário.

PRÉ-CONDIÇÕES
Não há

FLUXO BÁSICO
1. O usuário clica na aba “*Preview*”.
2. O sistema coleta as informações da Interface Abstrata e Concreta
3. Renderiza os elementos no protótipo segundo a **RN1**, **RN2**.
4. O caso de uso é encerrado.

FLUXOS ALTERNATIVOS
Não há.

REGRAS DE NEGÓCIO
(**RN1**) Se um elemento possui dados associados à ele, seus filhos devem ser renderizados de acordo com o tamanho do conjunto de dados.

(**RN2**) Se um *Bind* está associado um elemento, na renderização, o valor deve ser buscado na Interface Abstrata.

4 – Projeto do Programa

O objetivo desse capítulo é mostrar como foi projetada o programa e como está a distribuição de arquivos, afim de facilitar o entendimento e localização de alguma funcionalidade no código fonte.

4.1 – Diagrama de Classe

O diagrama de classes do Mira-Editor visa definir de forma clara as relações entre as classes do sistema. O primeiro ponto de destaque do diagrama fica herança de um *HTMLElement*, que representa um elemento concreto do MIRA. Através dessa herança, podemos criar diversos tipos de elemento concretos, sem que eles percam as características comuns aos outros tipos de elementos concretos.

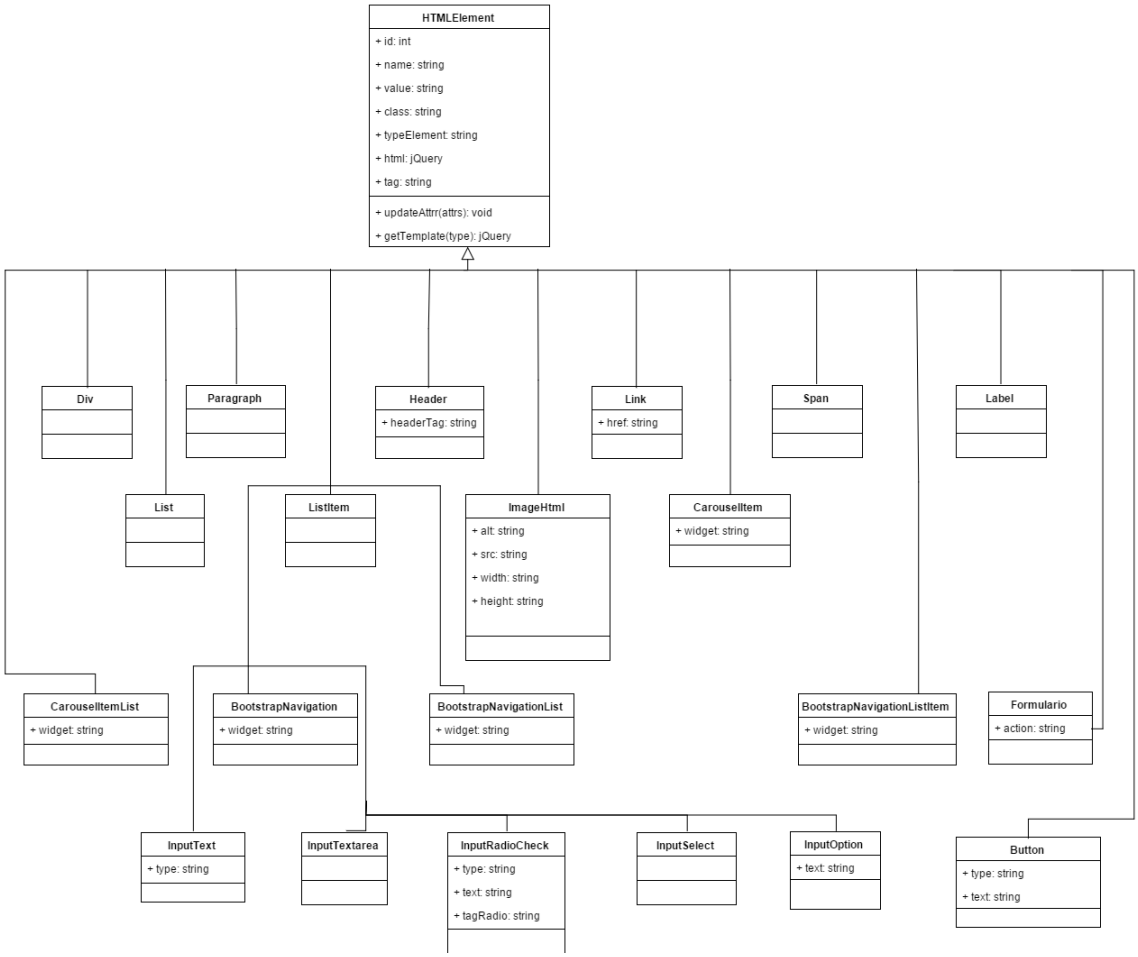

Figura 8 - Herança para elementos concretos.

A relação entre Interface Abstrata com elementos abstratos e Interface Concreta com elementos concretos também é visível no diagrama. A notação de

composição na relação entre elementos abstratos (*AbstractElement*) mostra uma das regras de negócio do caso de uso “Remover elemento”. Se um item é removido, seus filhos também serão removidos.

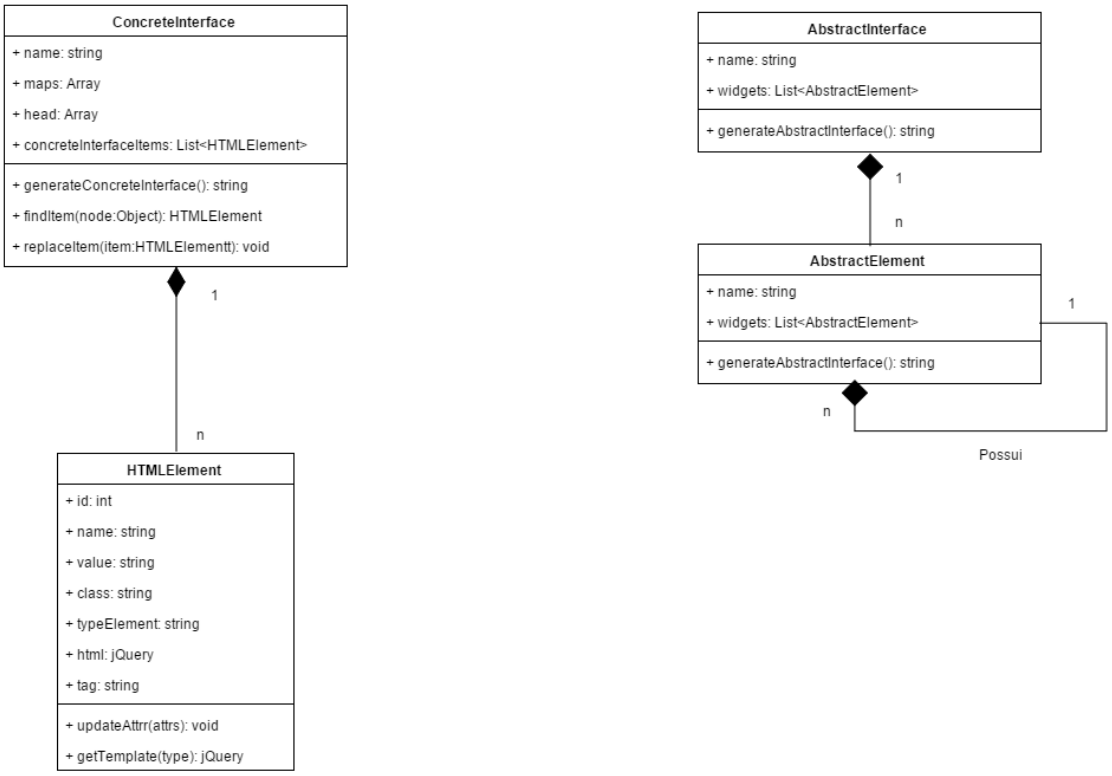

Figura 9 - Relação entre entidades.

Como já dito nesse trabalho, o Mira-Editor foi apoiado por algumas bibliotecas, plug-ins e frameworks. O diagrama de classe nos mostra a relação de uso classes do projeto com esses apoiadores e a relação entre elas também, pois uma pode utilizar outra para executar alguma determinada operação que não é de seu escopo.

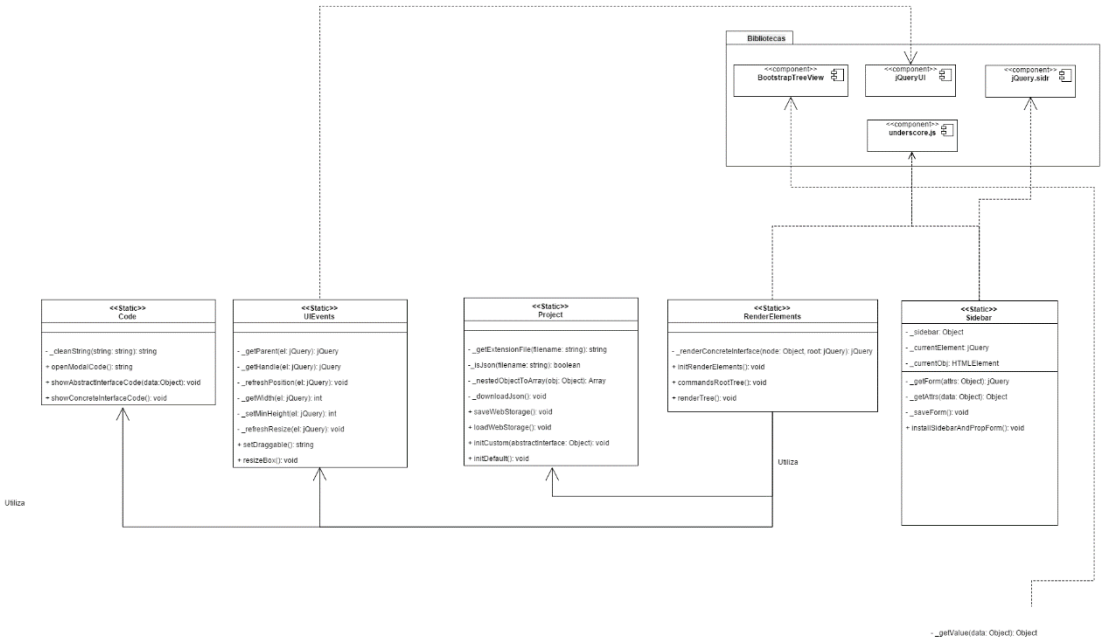

Figura 10 - Relação entre classes e as bibliotecas de apoio.

O diagrama de classes completo pode ser visto na imagem **diagrama_classe.png**, dentro da pasta "**img/diagramas**".

4.2 – Pastas e Arquivos

A seguir segue a lista descrevendo as pastas e arquivos importantes do projeto:

- **css**: Pasta contendo os estilos utilizados no projeto.
- **docs**: Pasta contendo o conteúdo de documentação das classes. Todo o conteúdo dessa pasta é gerado pela API de documentação *YUIdoc*[11].
- **fonts**: Fontes usadas pelo *Font Awesone*.
- **img**: repositório de imagens da aplicação.
- **js**: pasta contendo os arquivos javascript.
    - **lib**: pasta com as bibliotecas e plug-ins que não sofreram alterações.
    - **Code.js**: possui a classe Code e métodos de testes para geração de código das interfaces.
    - **Entities.js**: possui as classes referentes a Interface Abstrata, Concreta, elemento abstrato e concreto.
    - **Globals.js**: possui as variáveis globais do sistema.
    - Init.js: responsável por inicializar a aplicação.
    - **Preview.js**: Classe para controle a visualização do Preview.
    - **Project.js**: Classe de controle do projeto.
    - **RenderElements.js**: Classe que controla a exibição da Interface Abstrata e Concreta.

    - **Sidebar.js**: Classe para controle do formulário de propriedade dos elementos concretos.
    - **Tree.js**: plug-in Bootstrap-treeview, com alterações voltadas para construção de gerenciamentos da *treeview*.
    - **yuidoc.json**: Configuração para geração de documentação
- **index.html**: Página inicial da aplicação.
- **MiraEditorGuiaUsuario.pdf**: Manual do usuário.
- **templates.html**: Arquivo contendo o templates dos elementos e dos formulários de propriedades que serão usados pelo *Underscore.js*.

4.3 - Documentação

A documentação das classes pode ser encontrada no link http://tecweblab.github.io/mira-editor/docs/. Nessa página é possível consultar as classes, atributos e eventos presente no Mira-Editor.

5 – Testes

Os testes foram empregados para se adequar aos casos de uso já falados nesse trabalho. O objetivo principal foi atender os requisitos de cada caso de uso, dando uma atenção especial as regras de negócio, testando assim métodos no sistema que influenciavam diretamente nisso.

Os testes foram feitos automaticamente por um *script* com o auxílio da biblioteca *QUnit*[12] e para executá-lo basta clicar no menu “Testes”, no canto superior esquerdo da aplicação. As informações serão apresentadas para o usuário ao fim do processo e um arquivo com os logs dos testes será baixado.

5.1 – Teste: Salvar projeto

OBJETIVO
Verificar se o projeto foi salvo com os valores corretos.

PRÉ-REQUISITOS
Não há

PASSOS
1. Salvar os dados no *localStorage*.
2. Buscar os dados no *localStorage*.
3. Verificar se os resultados são os mesmos.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Dados salvos e captados no *localStorage* iguais.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Projeto: Salvar projeto”**.

5.2 – Teste: Novo projeto

OBJETIVO
Verificar se foi iniciado um projeto com os valores padrões.

PRÉ-REQUISITOS
Não há

PASSOS
1. Inicializar o projeto com os valores padrões.
2. Verificar se a Interface Concreta, Abstrata e o nome do projeto estão com os valores padrões.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Os dados do projeto devem ser os valores padrões.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Projeto: Novo projeto”**.

5.3 – Teste: Abrir projeto

OBJETIVO
Verificar se o projeto foi iniciado com os valores contidos no arquivo.

PRÉ-REQUISITOS
Não há.

PASSOS
1. Verificar a extensão de um arquivo.
2. Verificar se o arquivo é válido.
3. Verificar se os dados foram carregados com sucesso.

DADOS DE ENTRADA
Um arquivo de entrada.

RESULTADO ESPERADO
Dados projeto iguais aos dados contidos no arquivo de entrada.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Projeto: Novo projeto”**.

5.4 – Teste: Adicionar item à raiz da árvore

OBJETIVO
Verificar a adição de elementos na raiz da árvore.

PRÉ-REQUISITOS
Não há.

PASSOS
1. Clicar no botão e pressionar **Enter** no campo.
2. Clicar no botão e pressionar **Tab** no campo.
3. Clicar no botão e pressionar **Esc** no campo.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Ao pressionar Tab ou Enter a operação de adição é confirmada, porém ao pressionar Esc, a operação é cancelada.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **"Interface Abstrata: Adicionar item à raiz da árvore"**.

5.5 – Teste: Remover itens selecionado da raiz

OBJETIVO
Verificar se os itens selecionados da raiz estão sendo removidos.

PRÉ-REQUISITOS
Ter mais de um item na árvore.

PASSOS
1. Selecionar os itens da raiz que deseja remover.
2. Clicar no botão de remoção.
3. Verificar se os itens foram removidos.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Os itens selecionados da raiz devem ser removidos.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **"Interface Abstrata: Remover itens selecionados da raiz"**.

5.6 – Teste: Editar elemento

OBJETIVO
Verificar se a edição de um elemento é feita com sucesso levando em conta a tecla de confirmação.

PRÉ-REQUISITOS
Ter pelo menos um item na árvore.

PASSOS
1. Clica no botão editar de um elemento.
2. Pressionar a tecla Enter no campo e verificar se foi confirmada a operação.
3. Repetir a operação pressionando a tecla Tab e verificar se a mesma foi confirmada.
4. Repetir a operação pressionando a tecla Esc e verificar se a mesma foi cancelada.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Ao pressionar Tab ou Enter a operação é confirmada e cancelada ao pressionar Esc.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Abstrata: Editar elemento”**.

5.7 – Teste: Adicionar filho

OBJETIVO
Verificar a adição de um item para um nó existente.

PRÉ-REQUISITOS
Ter pelo menos um item na árvore.

PASSOS
1. Clica no botão adicionar filho de um elemento.
2. Pressionar a tecla Enter no campo e verificar se foi confirmada a operação.
3. Repetir a operação pressionando a tecla Tab e verificar se a mesma foi confirmada.
4. Repetir a operação pressionando a tecla Esc e verificar se a mesma foi cancelada.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO

Ao pressionar Tab ou Enter a operação é confirmada e cancelada ao pressionar Esc.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Abstrata: Adicionar filho”**.

5.8 – Teste: Remover elemento

OBJETIVO
Verificar se, ao excluir um item, seus filhos também são removidos.

PRÉ-REQUISITOS
Ter pelo menos um item na árvore com filhos.

PASSOS
1. Clicar no botão de remoção de um elemento.
2. Verificar se os filhos foram removidos.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Ao pressionar Tab ou Enter a operação é confirmada e cancelada ao pressionar Esc.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Abstrata: Remover elemento”**.

5.9 – Teste: Remover filhos selecionados

OBJETIVO
Verificar a remoção dos itens selecionados de um nó existente.

PRÉ-REQUISITOS
Ter pelo menos um item na árvore com filhos.

PASSOS
1. Selecionar os itens a serem removidos.
2. Clicar no botão de remover filhos selecionados do pai.
3. Verificar se os itens foram removidos.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Ao pressionar Tab ou Enter a operação é confirmada e cancelada ao pressionar Esc.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Abstrata: Remover filhos selecionados”**.

5.10 – Teste: Inserir Dados

OBJETIVO
Verificar se a requisição e passagem dos dados é feita com sucesso.

PRÉ-REQUISITOS
Ter pelo menos um item na árvore com filhos.

PASSOS
1. Selecionar os itens da árvore com filhos.
2. Fazer uma requisição *AJAX* com uma *url*/expressão inválida.
3. Fazer uma requisição *AJAX* com uma *url*/expressão válida.
4. Verificar se os dados foram passados para os filhos.

DADOS DE ENTRADA
Uma *url*/expressão válida e outra inválida.

RESULTADO ESPERADO
Que a requisição falhe para a *url*/expressão inválida e que os dados sejam passados para os filhos na requisição válida.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Abstrata: Inserir dados”**.

5.11 – Teste: Alterar tipo do elemento

OBJETIVO
Verificar se a operação de mudança do elemento é feita com sucesso.

PRÉ-REQUISITOS
Ter pelo menos um elemento concreto.

PASSOS
1. Alterar o tipo do elemento.
2. Verificar se a operação foi feita com sucesso.

DADOS DE ENTRADA
Uma *url*/expressão válida e outra inválida.

RESULTADO ESPERADO
Que a requisição falhe para a *url*/expressão inválida e que os dados sejam passados para os filhos na requisição válida.

RESULTADO OBITIDO

Consultar os logs gerados na sessão **“Interface Concreta: Alterar tipo do elemento”**.

5.12 – Teste: Arrastar elemento

OBJETIVO
Verificar se, ao arrastar um elemento, o mesmo respeita as regras de negócio.

PRÉ-REQUISITOS
Ter pelo menos um elemento concreto com filhos.

PASSOS
1. Identificar o pai do elemento concreto.
2. Arrastar o elemento concreto.
3. Verificar se ele não saiu do interior do elemento pai.

DADOS DE ENTRADA
Não há

RESULTADO ESPERADO
Que o elemento não saia do limite do elemento pai.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Concreta: Arrastar elemento”**.

5.13 – Teste: Redimensionar elemento

OBJETIVO
Verificar se, ao redimensionar um elemento, o mesmo respeita as regras de negócio.

PRÉ-REQUISITOS
Ter pelo menos um elemento concreto com filhos.

PASSOS
1. Redimensionar e verificar se não está menor que a largura mínima.
2. Redimensionar e verificar se a altura é superior a soma da altura dos filhos.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Não ter largura menor que 200px e não ser maior que a altura mínima, segundo a regra de negócio

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Concreta: Redimensionar elemento”**.

5.14 – Teste: Propriedades do elemento

OBJETIVO
Verificar se, ao abrir o formulário de propriedades, o campo “valor” está preenchido com “$bind”, para um elemento abstrato com *Bind* associado.

PRÉ-REQUISITOS
Ter pelo menos um elemento abstrato com um *Bind* associado.

PASSOS
1. Clicar no botão de propriedades do elemento.
2. Verificar se o campo “Valor” está com o valor “$bind”.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Que abra o formulário de propriedades e que o campo “Valor” esteja com o valor “$bind”.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Interface Concreta: Propriedades do elemento”**.

5.15 – Teste: Salvar propriedades

OBJETIVO
Verificar se o método *updateAttr* está funcionando corretamente.

PRÉ-REQUISITOS
Não há.

PASSOS
1. Executar o método *updateAttr* com um conjunto de atributos como parâmetro.

DADOS DE ENTRADA
Não há.

RESULTADO ESPERADO
Que o elemento atualizar os valores dos atributos passados como parâmetro.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **“Propriedades: Salvar propriedades”**.

5.16 – Teste: Gerar código das interfaces

OBJETIVO
Verificar se a geração do código para a Interface Concreta e Abstrata está de acordo com o esperado.

PRÉ-REQUISITOS
Não há

PASSOS
1. Gerar o código das Interfaces.
2. Verificar se está de acordo com o esperado.

DADOS DE ENTRADA
Modelo com os elementos abstratos e concretos.

RESULTADO ESPERADO
Que o código gerado esteja de acordo com o esperado.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **"Código Gerado: Gerar código das interfaces"**.

5.17 – Teste: Exibir *Preview*

OBJETIVO
Verificar se está captando os valores de forma correta.

PRÉ-REQUISITOS
Não há

PASSOS
1. Testar o método *_getValue*.
2. Verificar se o valor captado é o esperado.

DADOS DE ENTRADA
Modelo com os elementos abstratos e concretos.

RESULTADO ESPERADO
Que o código gerado esteja de acordo com o esperado.

RESULTADO OBITIDO
Consultar os logs gerados na sessão **"*Preview*: Exibir *Preview*"**.

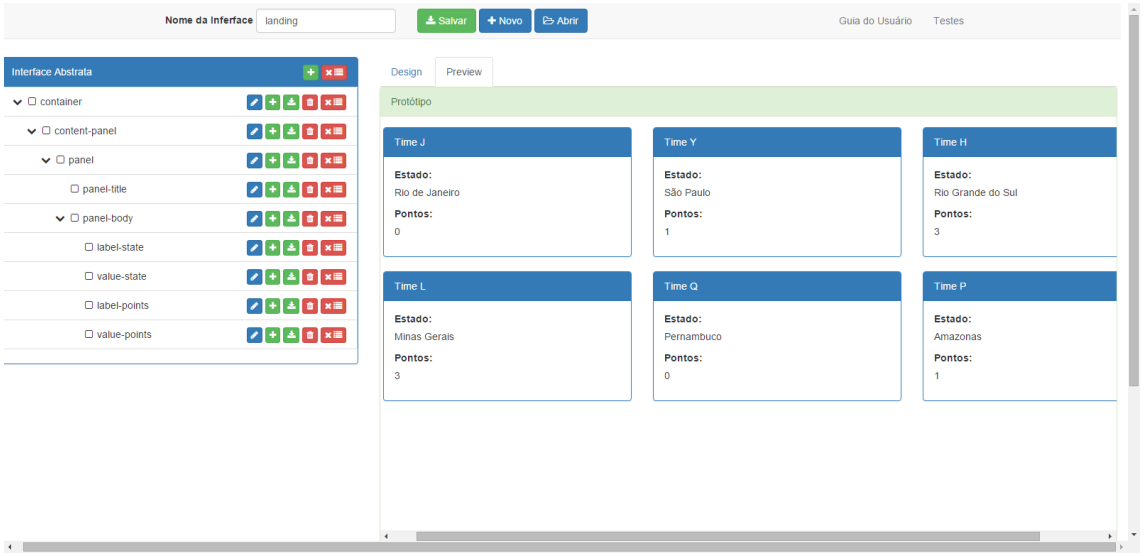

Figura 11 - Preview gerado pelos testes.

6 – Trabalhos Futuros

Os planos para o futuro é deixar a Mira-Editor com uma interface similar à de uma IDE, onde os usuários poderão arrastar elementos para o protótipo e trabalha-lo da maneira que deseja e de forma mais fácil. Outra funcionalidade importante a ser implementada é a utilização de regras de seleção de elementos abstratos e concretos. Dessa forma, podemos fazer com que um determinado elemento possa ser representado de diversas maneiras, de acordo uma determinada regra. Essa é uma das grandes funcionalidades do Mira.

7 – Referências

[1] https://github.com/TecWebLab/mira
[2] http://www.json.org/
[3] https://jquery.com/
[4] http://www.w3schools.com/js/js_htmldom.asp
[5] http://james.padolsey.com/javascript/cross-domain-requests-with-jquery/
[6] https://jqueryui.com/
[7] http://underscorejs.org/
[8] http://getbootstrap.com/
[9] http://www.jondmiles.com/bootstrap-treeview
[10] https://github.com/lydell/json-stringify-pretty-compact
[11] http://yui.github.io/yuidoc/
[12] https://qunitjs.com/