
rung-cli Documentation

Release rst-ols

Mar 14, 2018

Contents

1	Definição	3
1.1	O que é o Rung	3
2	Características	5
2.1	Contexto	5
2.2	Chaves primárias	6
2.3	Extensões assíncronas	6
2.4	Tipos de parâmetros	7
2.5	Banco de dados	9
2.6	Internacionalização e tradução	9
2.7	Cards personalizados	10
2.8	Hot reloading	11
2.9	Barra lateral personalizada	13
2.10	Ícone personalizado	15
2.11	Rung Bot	15
2.12	Resources	16
2.13	AutoComplete	18
2.14	Informações adicionais	19
3	Rung CLI	21
3.1	Compilando uma extensão	21
3.2	Executando uma extensão	21
3.3	Publicação	22
3.4	Boilerplate	22
3.5	Documentação da extensão	23
3.6	Banco de dados	23
4	Extras	25
4.1	Extraindo HTML	25
4.2	Resolução de problemas	26
5	Ajuda ou parceiras?	27



Aprenda a desenvolver extensões para o [Rung](#).

1.1 O que é o Rung

O Rung é a primeira ferramenta do mundo para gestão de excepcionalidades. O objetivo da ferramenta é te alertar de situações que podem ser importantes para você ou para sua empresa. O Rung é altamente customizável. Há uma loja de extensões que permite a você configurar o que quer monitorar, e também lhe permite programar suas próprias extensões.

1.1.1 O que é uma extensão

Uma extensão é um aplicativo para o Rung, que foi previamente programado para receber uma série de parâmetros e, quando houver a satisfação de uma condição específica, gerar alertas e notificar o usuário sobre. Um exemplo simples seria a cotação do dólar. Por exemplo, eu posso querer ser avisado quando o dólar estiver abaixo de R\$ 3,00.

2.1 Contexto

O contexto é o primeiro parâmetro passado para a função principal da sua extensão. Ele traz informações importantes, como o locale do usuário e os valores dos parâmetros que são informados.

Exemplo:

```
import { create } from 'rung-sdk';
import { Integer } from 'rung-sdk/dist/types';

function main(context) {
  const { a, b } = context.params;

  return {
    alerts: [`Sum of ${a} and ${b} is ${a + b}`]
  };
}

const params = {
  a: {
    description: 'First number',
    type: Integer
  },
  b: {
    description: 'Second number',
    type: Integer
  }
};

export default create(main, { params });
```

O contexto também é responsável por carregar as informações do banco de dados de cada instância de extensão, em `context.db`.

2.2 Chaves primárias

Cada alerta pode ter uma chave primária. A chave primária serve para localizar aquele alerta e o preservar em uma atualização. É indicado que todo alerta possua uma chave primária para que seus metadados não sejam perdidos durante cada execução da extensão.

Para habilitar o suporte às chaves primárias, deve-se utilizar o segundo parâmetro da função `create` da SDK do Rung:

```
export default create(main, {
  params,
  primaryKey: true
});
```

Por padrão, os alertas devem ser retornados em um objeto contendo `{ alerts }`. Nessa situação, `alerts` deve conter um objeto ou *array* com as chaves e os valores de cada alerta. No caso de um *array*, a chave utilizada será o índice. Com ES6, você pode construir um objeto com chaves dinâmicas utilizando a sintaxe `{ [key]: value }`. Exemplo:

```
import { map, mergeAll } from 'ramda';

function render({ id, name, score }) {
  return { [id]: `${name} with ${score} points!` };
}

function main(context) {
  const students = [
    { id: '0001', name: 'Bianca del Rio', score: 8 },
    { id: '0002', name: 'Violet Chachki', score: 9 },
    { id: '0003', name: 'Dida Ritz', score: 3 }
  ];

  return {
    alerts: mergeAll(map(render, students))
  };
}

export default create(main, { params: {}, primaryKey: true });
```

2.3 Extensões assíncronas

O Rung possui dois tipos de chamada de retorno para a função principal. Uma extensão pode ser síncrona ou assíncrona. Uma extensão síncrona executa linha a linha e tem seu valor processado retornado. É o caso, por exemplo, de uma extensão que parabeneze o indivíduo no seu aniversário. Quando uma extensão precisa “esperar” uma resposta, temos o caso de uma extensão assíncrona. Exemplos são requisições HTTP.

Para diferenciar extensões síncronas e assíncronas, utilizamos um segundo parâmetro na função principal, o `done`. Quando o `done` é passado, a extensão não termina de executar até que a função `done` seja chamada com os alertas a serem gerados. Na outra situação, a extensão termina sua execução até quando o primeiro `return` for encontrado.

Extensão síncrona

```
function main(context) {
  return {
    alerts: ["Hello world!"]
  };
}
```

```
}
}
```

Extensão assíncrona

```
function main(context, done) {
  setTimeout(() => {
    done({
      alerts: ["Hello world!"]
    });
  }, 3000);
}
```

No segundo exemplo, a extensão leva 3 segundos para executar e então gerar o alerta Hello world!.

2.4 Tipos de parâmetros

Os parâmetros de entrada de uma extensão do Rung possuem diversos tipos com múltiplas finalidades, que variam de acordo com como você vai trabalhar com o dado e como você deseja que ele seja renderizado para o usuário durante a instalação. Os tipos podem ser importados via:

```
import {} from 'rung-cli/dist/types';
```

2.4.1 Integer

Para números inteiros tendendo de negativo a infinito positivo, a depender da representação de números inteiros em bits de onde está rodando.

2.4.2 Double

Para números de ponto flutuante (decimais), a depender da representação de números de ponto flutuante em bits de onde está rodando, geralmente segundo a [IEEE754](#).

2.4.3 DateTime

Indisponível

2.4.4 Natural

Números naturais de 0 a infinito. Não são aceitos números negativos

2.4.5 Char(n)

Textos de até n caracteres, onde n corresponde a um número natural. Se tiver como entrada um texto maior, este é cortado até o limite.

2.4.6 IntegerRange(m, n)

Intervalos numéricos de números inteiros entre m e n . É possível, por exemplo, simular números naturais com `IntegerRange(0, Infinity)`. m sempre precisa ser menor do que n para que haja uma situação válida.

2.4.7 DoubleRange(m, n)

Intervalos numéricos de números de ponto flutuante (decimais) entre m e n , onde m precisa ser um valor menor que n .

2.4.8 Money

Indisponível

2.4.9 String

Qualquer trecho de texto arbitrário

2.4.10 Color

Cor em hexadecimal, começando por #.

2.4.11 Email

Qualquer email válido.

2.4.12 Checkbox

Indisponível

2.4.13 OneOf(xs)

Qualquer elemento de xs , onde xs deve ser do tipo `string[]` (lista de strings).

2.4.14 Url

Url independente de protocolo (HTTP, FTP ...).

2.4.15 IntegerMultiRange(m, n)

Slider em que 2 pontos são retornados, estão esses pontos entre m e n .

2.4.16 Calendar

Para datas, com renderização de um calendário.

2.4.17 AutoComplete

Para campos que podem ser interdependentes, em modo texto e ter um arquivo expondo uma função de *autocomplete*.

2.4.18 Location

Qualquer trecho de texto que faça referência a uma localização (cidade, estabelecimento, país, etc). Quando o campo for utilizado no [Rung](#), irá ser carregado um autocomplete com sugestões de locais vindas do [Google](#).

2.5 Banco de dados

Também há suporte para micro esquema de banco de dados não-relacional. Cada extensão pode armazenar um objeto JS que possa ser representado como JSON. Esse objeto pode ser retornado juntamente com os alertas da extensão. Durante a primeira execução, o valor sempre corresponde a *undefined*.

Exemplo de contador

```
import { create } from 'rung-sdk';

function main(context) {
  const counter = context.db === undefined ? 0 : context.db;

  return {
    alerts: [`The value is ${counter}`],
    db: counter + 1
  };
}

export default create(main, { params: {}, primaryKey: true });
```

Cada vez que a extensão roda, o contador é incrementado. Caso ele seja *undefined*, é inicializado como 0. Você pode colocar objetos de complexidade bem maiores dentro dele.

Para mais informações sobre a utilização com o Rung CLI, visite [db](#).

2.6 Internacionalização e tradução

O Rung tem suporte a múltiplos idiomas nas extensões. Os locales do Rung se baseiam na combinação por `_` dos formatos ISO 639 com o código do país, especificado na ISO 3166. Exemplos são `en_US` e `pt_BR`.

Há uma função global chamada `_`, que recebe um texto a ser internacionalizado, preferencialmente em inglês, e opcionalmente um conjunto de pares a interpolar. O Rung buscará por arquivos `.json` com o nome do locale dentro da pasta `locales/`. Exemplo:

locales/pt_BR.json

```
{
  "Hello world!": "Olá, mundo!"
}
```

locales/zh_CN.json

```
{
  "Hello world!": ""
}
```

index.js

```
function main(context) {
  return {
    alerts: [_( "hello world!") ]
  };
}
```

Caso o locale não for encontrado, a chave é utilizada, no caso, a versão em inglês no exemplo.

2.6.1 Interpolação de strings

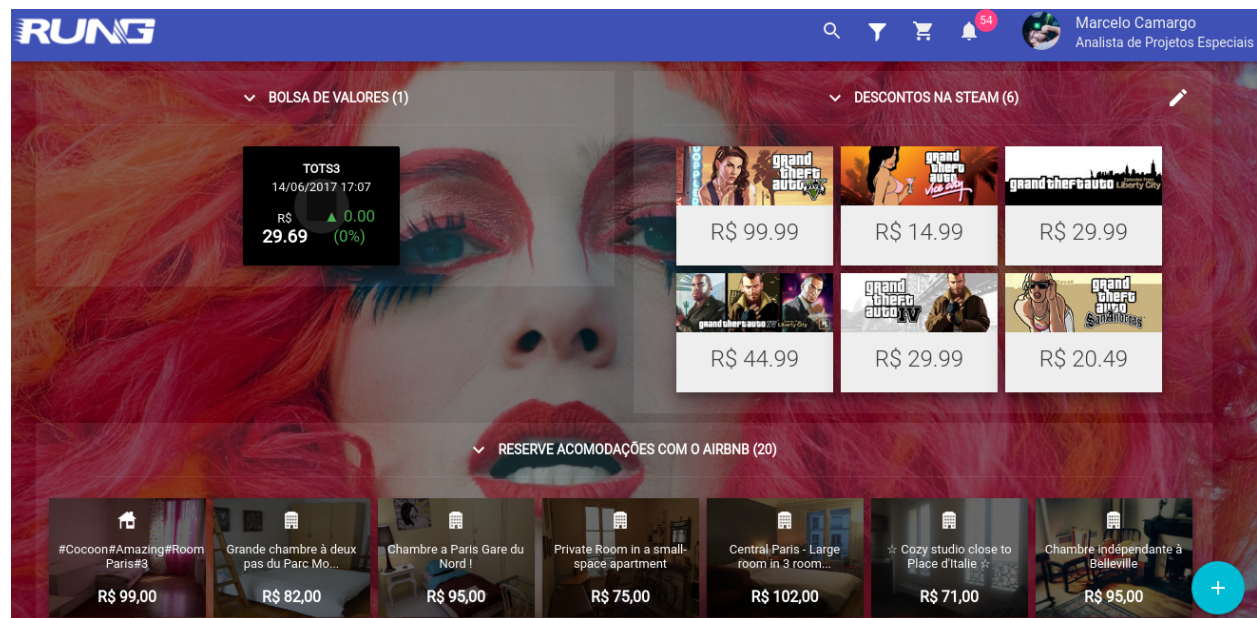
A função `_` aceita um segundo parâmetro para interpolar, tornando possível fazer, por exemplo:

```
return {
  alerts: [_( "good morning, {{name}}!", { name: 'Trixie Mattel' })]
};
```

Note que, no arquivo `.json`, os nomes dentro das chavetas devem ser **preservados**

2.7 Cards personalizados

Além de texto plano, o Rung permite que os cards possam ter um subset de HTML personalizado para cada extensão. O compilador de extensões do Rung possui suporte direto a JSX, permitindo que o conceito do React seja aplicado aos cards em tempo de compilação.



2.7.1 JSX

Ao invés de definir o HTML como string, utilize-o diretamente na renderização. Também há suporte a folha de estilos como objetos dentro das tags Exemplo de extensão customizada:

```
import { create } from 'rung-sdk';

const styles = {
  name: {
    fontWeight: 'bold',
    color: 'red'
  }
};

function render(name) {
  return (
    <div>
      Hello, <span style={ styles.name }>{ name }</span>
    </div>
  );
}

function main(context) {
  return {
    alerts: [{
      title: 'Betty',
      content: render('Betty')
    }]
  };
}

export default create(main, { params: {}, primaryKey: true });
```

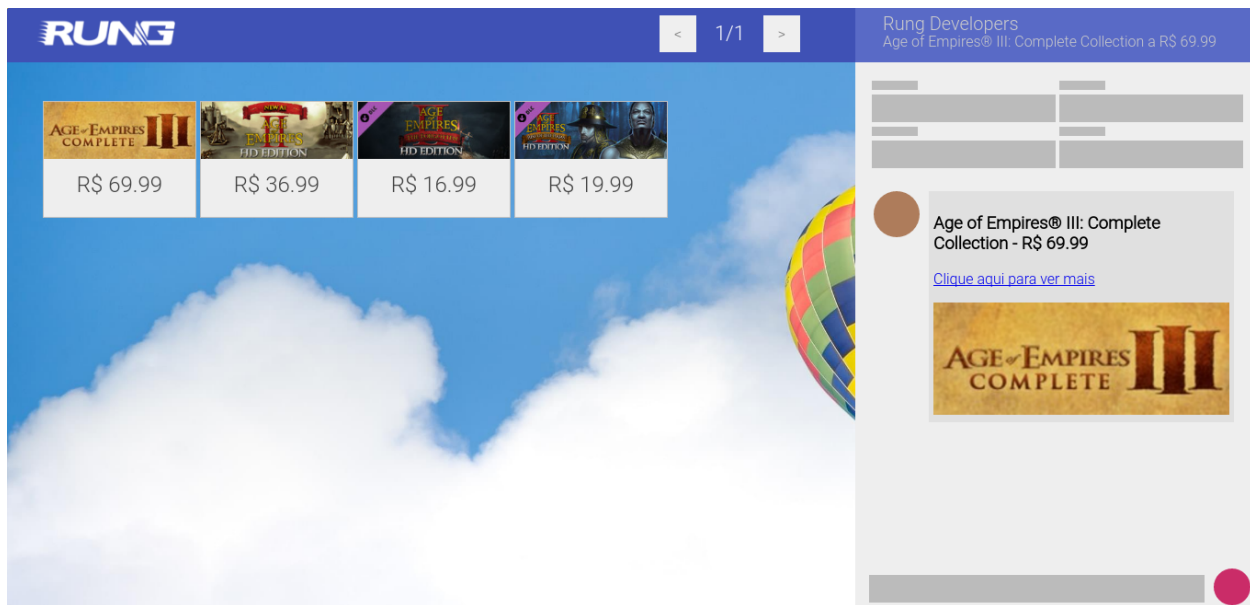
2.7.2 Preview

É ideal especificar como o card será exibido para o usuário durante a instalação da extensão dentro do Rung App. Para isso, você pode definir nas configurações o parâmetro `preview` e reutilizar a função `render`:

```
-export default create(main, { params: {}, primaryKey: true });
+export default create(main, {
+  params: {},
+  primaryKey: true,
+  preview: render('Salette')
+});
```

2.8 Hot reloading

A partir da versão 1.1.1, o Rung CLI suporta desenvolvimento agilizado através de edição ao vivo e compilação dinâmica. Se você está habituado a desenvolver com React, vai se sentir familiarizado.



2.8.1 Como configurar

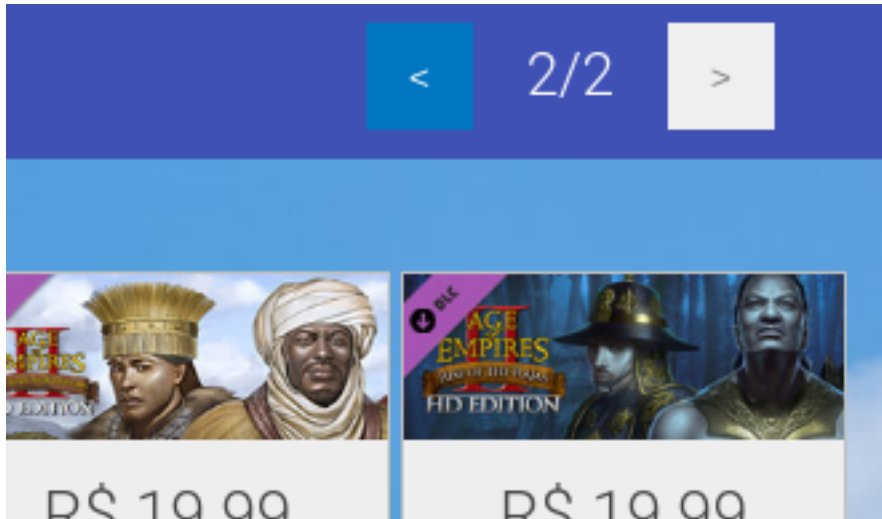
Para habilitar o modo `live`, rode sua extensão com `rung run --live`, e preencha os valores dos parâmetros de entrada. Nada mais é necessário. O Rung CLI irá monitorar as mudanças e compilar dinamicamente!

```
~/p/e/steam-store ➤ master ± ➤ rung run --live
? Nome do item a ser procurado age of empires
? Desconto mínimo desejado, em % 0
? Preço máximo desejado 100
i Info:
  _ _ _ _ _
 / _ \ _ _ / | / / _
 / , _ / / / / ( /
/_/ | \ _ , _ / | \ _ _

i Info: new session for ::1
i Info: changes detected. Recompiling...
✓ Success: wow! recompiled and executed in 992ms!
i Info: changes detected. Recompiling...
✓ Success: wow! recompiled and executed in 787ms!
```

2.8.2 Linha do tempo

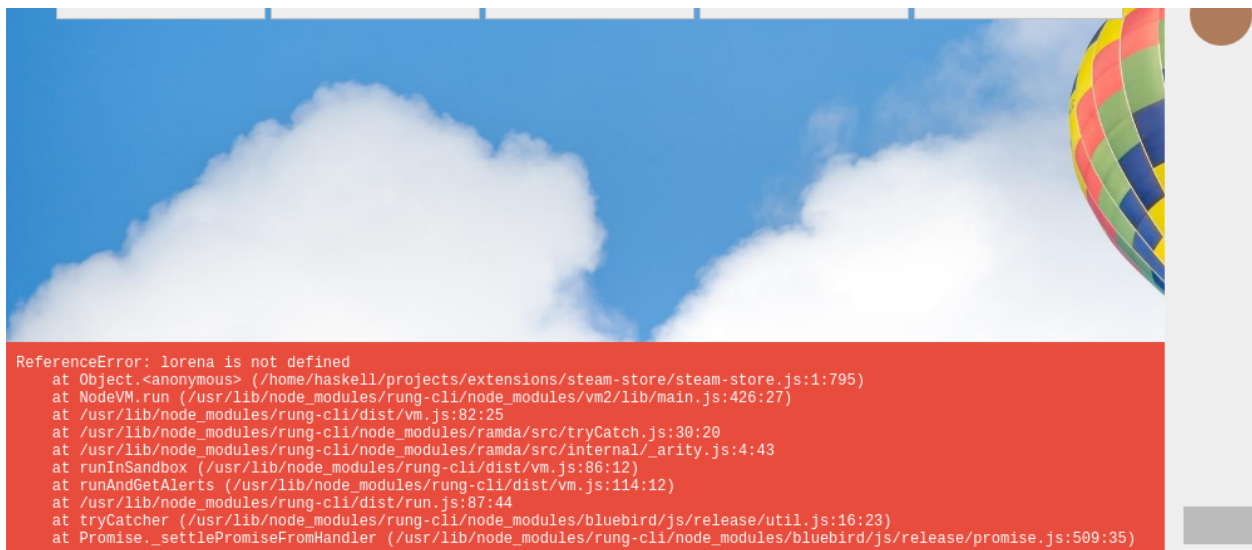
Como o estado é puro, uma linha do tempo é criada e você pode navegar pelos estados anteriores. Quando uma mudança é detectada, uma nova era é criada e a aplicação é reposicionada para ela.



Caso você interrompa o processo e entre com novos parâmetros, não é necessário fechar o navegador ou recarregar a página. O mesmo é válido para quando executar outra extensão. As novas entradas serão adicionadas à linha do tempo das anteriores.

2.8.3 Erros

Erros sintáticos e de *runtime* serão exibidos em uma caixa específica.



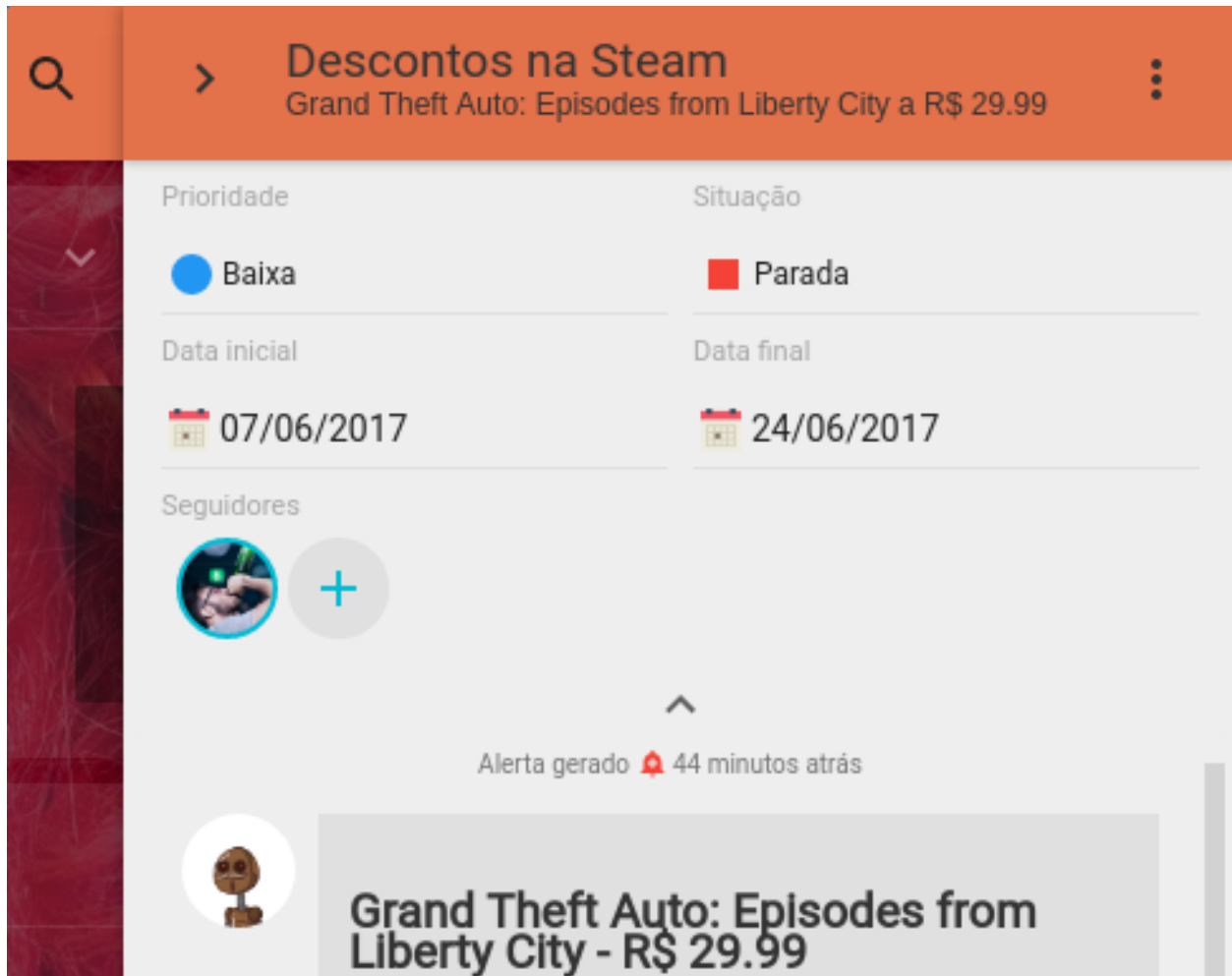
2.8.4 Barra lateral

Você pode clicar nos alertas para ver o resultado da renderização do *markdown* para comentários na barra lateral.

2.9 Barra lateral personalizada

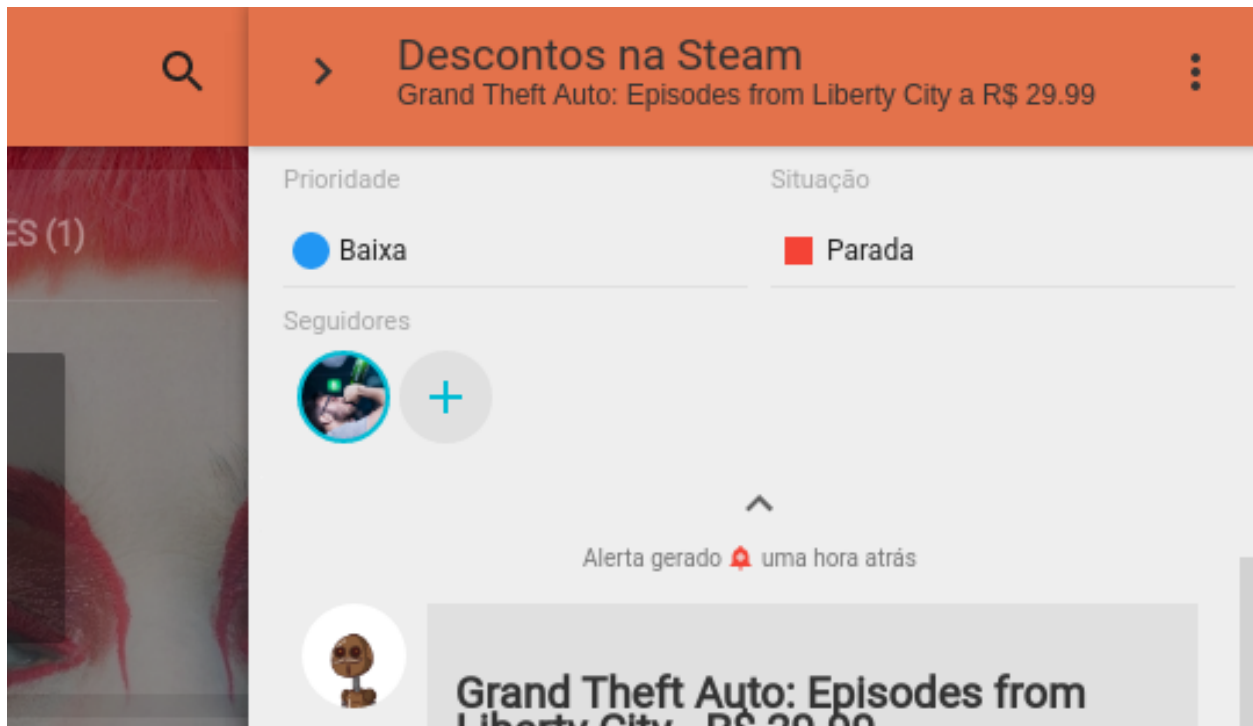
Usando a propriedade `sidebar` nas configurações, é possível escolher quais campos deseja ocultar. Por padrão, a barra lateral possui suporte a 4 campos:

Identificador	Descrição
priority	Prioridade
situation	Situação
startDate	Data inicial
endDate	Data final



Se quisermos, por exemplo, remover os campos de data:

```
export default create(main, {
  params,
  primaryKey: true,
  sidebar: {
    startDate: false,
    endDate: false
  }
});
```



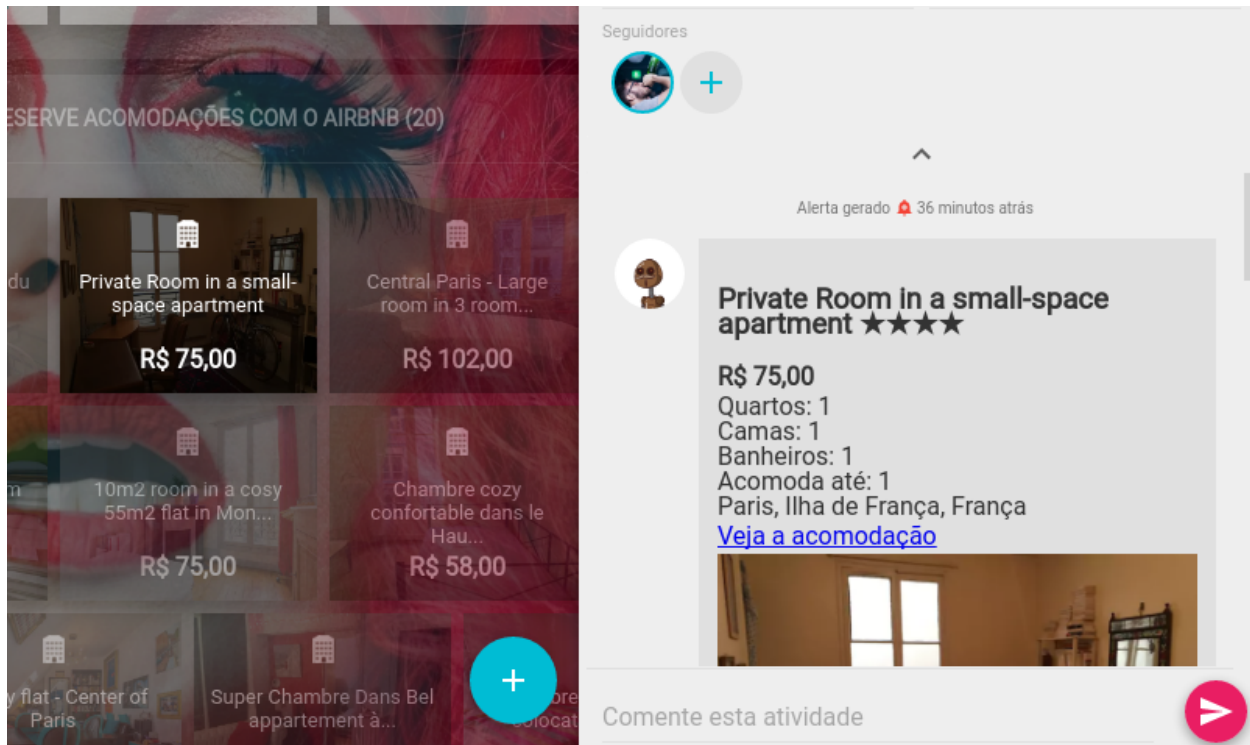
2.10 Ícone personalizado

Ao publicar uma extensão, o Rung tentará localizar um arquivo `icon.png` no seu pacote. Ele corresponde à identidade de sua extensão. Não é obrigatório, mas é recomendado. Ele determina como sua extensão será mostrada na Rung Store:



2.11 Rung Bot

Em seus alertas, é possível programar para que haja um comentário no *follow-up* do Rung Bot, que é, basicamente, um robô customizado que tem o objetivo de complementar a informação do card.



O conteúdo escrito pelo Rung Bot deve ser definido usando Markdown, e paralelo ao card customizado. Exemplo:

```
return {
  alerts: [{
    title: 'Bananas are cheap',
    content: render(bananasPrice),
    comment: renderComment(bananasPrice)
  }]
};
```

```
function renderComment(bananasPrice) {
  return `
    # Bananas are cheap!
    The bananas are costing US$ ${bananasPrice}!!!
    Buy them now!
    ![Bananas] (https://bananas.org/banana.jpg)
  `;
}
```

2.12 Resources

Quando um alerta de uma integração possui uma ou várias imagens, pode-se utilizar a propriedade *resources*. Esta propriedade permite que as imagens sejam visualizadas em um único comentário e em forma de carrossel, deixando os comentários mais limpos e objetivos.



Reserve acomodações com o Airt

CAMPOS DO JORDÃO, SP, Brasil





Villa Paz - Vista Espetacular

★★★★★

R\$ 102,00
Quartos: 3
Camas: 10
Banheiros: 4
Acomoda até: 10
CAMPOS DO JORDÃO, SP, Brasil
[Veja a acomodação](#)

Por Rung Bot um dia atrás



Dê uma olhada nessas imagens



1 de 36

Por Rung Bot um dia atrás

Comente esta atividade



O conteúdo passado para a propriedade `resources` deve ser uma lista contendo as urls das imagens geradas pelo alerta. Exemplo:

```
return {
  alerts: [{
    title: 'Bananas are cheap',
    content: render(bananasPrice),
    comment: renderComment(bananasPrice),
    resources: [
      'http://www.bananas.com/banana1.jpg',
      'http://www.bananas.com/banana2.jpg',
      'http://www.bananas.com/banana3.jpg'
    ]
  }]
};
```

2.13 AutoComplete

O Rung suporta que os campos possam ter a função de autocompletar, provendo uma função JavaScript que lida com o retorno baseado na entrada.

Para que um campo seja autocompletável, é necessário defini-lo como um campo do tipo `AutoComplete`, disponível em `runcli/dist/types`.

Vamos criar um campo que tenha o autocompletar com nomes de Pokémons nesse exemplo!

- Defina o tipo do campo como `AutoComplete`

```
params: {
  pokemon: {
    type: AutoComplete,
    description: 'Pick a Pokémon!'
  }
}
```

- Crie um arquivo `autocomplete/pokemon.js`
- Exporte uma função que use a função de *callback* ou que retorne uma lista de strings

```
export default function ({ input, lib }) {
  return lib.request.get('https://raw.githubusercontent.com/BrunnerLivio/
  ↪PokemonDataGraber/master/output.json')
    .then(({ text }) => JSON.parse(text))
    .then(pokemons => pokemons.map(pokemon => pokemon.Name))
    .filter(name => name.startsWith(input));
}
```

E pronto, compile e suba sua extensão para o Rung! Você também pode testar essa funcionalidade na sua extensão diretamente do Rung CLI (automaticamente).

2.13.1 Parâmetros

A função exportada recebe como parâmetros:

1. Objeto contendo `input` (entrada do usuário) e `lib`

2. Callback opcional `done`. Se passado, deve ser chamado para retornar o controle. Senão, a função deve retornar uma `Promise`

2.13.2 Retorno

`Promise` ou chamada de `done` a um `Array<String>` contendo os dados já filtrados.

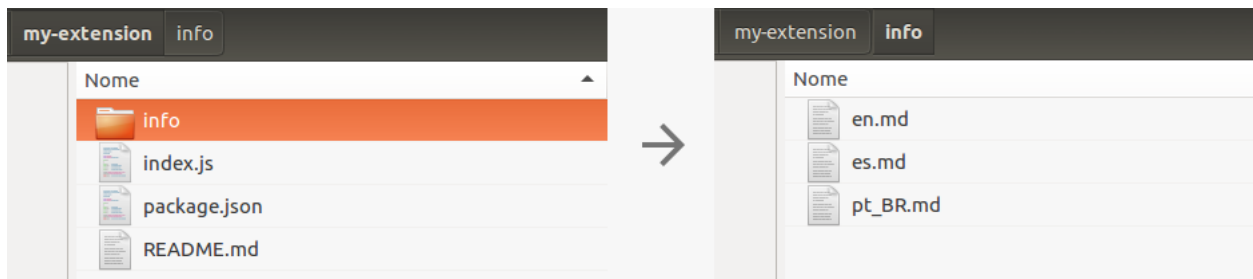
2.13.3 Bibliotecas

O parâmetro `lib` vindo dentro do primeiro objeto carrega duas bibliotecas, `ramda` e `request`. As requisições são feitas utilizando a biblioteca externa `superagent`.

2.14 Informações adicionais

Atualmente, o Rung possibilita que o desenvolvedor defina, além da descrição, um conteúdo adicional (no formato markdown) a ser apresentado na página da sua extensão na loja. Este conteúdo adicional é opcional, mas é recomendado o seu uso de tal forma que o seu conteúdo instigue o usuário a instalar e utilizar a extensão.

O conteúdo adicional deve ser estruturado dentro da pasta `info` e pode ser definido em inglês (`en`), português (`pt_BR`) e espanhol (`es`), conforme imagem abaixo.



3.1 Compilando uma extensão

Você pode usar o comando `rung build` para compilar uma extensão para a forma binária, gerando um arquivo `.rung`. O binário gerado pode então ser usado para distribuição e publicação para a loja do Rung, mas não precisa ser gerado para rodar uma extensão localmente, levando em conta que o *Rung CLI* consegue atuar diretamente como um interpretador.

O binário do Rung é uma forma modificada de um arquivo PKZip. Em tempo de compilação, a VM do Rung executa sua extensão para garantir que não há erros de análise do código exportado e gera os metadados para sua extensão de forma estática, para otimizar a execução.

3.2 Executando uma extensão

É possível testar as extensões que desenvolve localmente sem a necessidade de publicá-las para o Rung. O *Rung CLI* provê o comando `rung run` para isso.

```
~/p/e/steam-store ➤ master ➤ rung run
String> Nome do item a ser procurado: grand theft auto
IntegerRange(0, 100)> Desconto mínimo desejado, em %:
Double> Preço máximo desejado: 100
{ alerts:
  { '11902':
    { title: 'Grand Theft Auto: The Trilogy a R$ 44.99',
      content: '<div><div style="position:absolute;top:62px;height:
#424242;margin-top:14px">R$ 44.99</div></div></div>',
      comment: '\n          ### Grand Theft Auto: The Trilogy - *
sule_231x87.jpg?t=1450368998)](http://store.steampowered.com/app/11902)'
    },
    '12100':
    { title: 'Grand Theft Auto III a R$ 14.99',
      content: '<div><div style="position:absolute;top:62px;height:
#424242;margin-top:14px">R$ 14.99</div></div></div>',
      comment: '\n          ### Grand Theft Auto III - **R$ 14.99
```

É criada uma interface visual que recebe os parâmetros de entrada e mostra o resultado obtido, os alertas que seriam gerados, para o programador.

3.2.1 Executando com locale customizado

Quando estiver testando internacionalização, pode forçar a definição de um locale pelo próprio terminal, executando:

```
RUNG_LOCALE=pt_BR rung run
```

Por padrão, sem essa definição, o *Rung CLI* irá considerar o locale padrão do seu sistema operacional.

3.3 Publicação

Após o desenvolvimento, podemos publicar a extensão para a Rung Store. O Rung CLI implementa o comando `rung publish` para isso. Ao publicar uma extensão, informe seu usuário e senha do Rung.

Restrições de publicação

- A versão deve atender ao [semver](#);
- Ao atualizar uma extensão, a versão deve ser incrementada;
- É necessário possuir *karma* de desenvolvedor.

Para conseguir *karma* de publicação, entre em contato conosco em developer@rung.com.br.

Para publicar uma extensão, é obrigatório que `title` e `description` estejam definidos na configuração da extensão.

3.4 Boilerplate

O comando `rung boilerplate` pede algumas informações sobre a sua extensão e gera a base funcional de código para que você possa trabalhar. Os parâmetros perguntados são:

Pergunta	Descrição
Nome do projeto	Identificador único do projeto, em caixa-baixa e separado por -
Versão	Versão de acordo com o semver
Título	Título da sua extensão para exibição na Rung Store
Description	Descrição humana da proposta da sua extensão
Category	Identificador da categoria da sua extensão. Padrão: <code>miscellaneous</code>
Licença	Licença em que sua extensão é distribuída. Padrão MIT

Após confirmar, serão gerados `package.json`, `index.js` e `README.md` com as informações básicas de uma extensão.

3.5 Documentação da extensão

É possível usar o comando `rung readme` para gerar a documentação básica para uma extensão, incluindo informações sobre os parâmetros de entrada e dinâmicas ([exemplo](#)).

3.6 Banco de dados

Existem dois comandos no Rung CLI para trabalhar com banco de dados via linha de comando:

Comando	Descrição
<code>rung db clear</code>	Limpa toda a base de dados para a extensão ativa
<code>rung db read</code>	Permite visualizar em formato Yaml os dados gravados no banco



```

~ /p/e/buscape  master  rung db read
users:
-
  name:  Marcelo Haskell Camargo
  active: true
-
  name:  Paulo Henrique Cuchi
  active: true
-
  name:  Salete
  active: true
~ /p/e/buscape  master  |

```


4.1 Extraindo HTML

Esse exemplo visa mostrar a possibilidade de extrair o HTML do site que quer obter o conteúdo e manipular o DOM fictício para poder obter os dados em específico. Iremos criar uma extensão que gera alertas contendo os nomes de todas as drag queens do programa de TV americano *RuPaul's Drag Race*.

Estaremos usando o seguinte site como base: https://en.wikipedia.org/wiki/List_of_RuPaul%27s_Drag_Race_contestants

O conteúdo que queremos encontra-se em uma tabela no site:

Contestants [\[edit \]](#)

All information is accurate as of the time the season was filmed and thus may vary from season to season for returning contestants.

☐ ^ Contestant returned for a second time

Drag Name ↕	Real Name ↕	Age ¹ ↕	Hometown ↕	Season ↕	Finish ↕
Victoria "Porkchop" Parker	Victor Bowling	39	Raleigh, North Carolina	1	9th
Tammie Brown	Keith Glen Schubert	26	Los Angeles, California	1	8th
Akashia	Eric Flint	24	Cleveland, Ohio	1	7th
Jade	David Sotomayor ^[5]	25	Chicago, Illinois	1	6th
Ongina	Ryan Ong Palao	26	Los Angeles, California	1	5th
Shannel		26	Las Vegas, Nevada	1	4th
Rebecca Glasscock	Javier Rivera	26	Fort Lauderdale, Florida	1	3rd

É necessário conhecer o básico de seletores do DOM. Se você sabe CSS, você provavelmente tem o conhecimento necessário para selecionar os elementos estáticos.

A tabela que queremos possui uma classe `.sortable`. Nós queremos pegar, do corpo da tabela cada linha e, de cada linha, o primeiro elemento. Então, podemos montar nosso seletor como `.sortable tbody tr > td:nth-child(1)`. Se você abrir seu console e digitar `document.querySelectorAll('.sortable`

`tbody tr > td:nth-child(1) ')`, verá que recebemos uma lista de nodos HTML com as informações que queremos!

A requisição para obter HTML é igual à requisição para obter JSON. Podemos usar a *Superagent* para isso, e também vamos usar uma outra dependência. Adicione o `jsdom` ao projeto. Usaremos ele para criar o DOM virtual!

```
import Bluebird from 'bluebird';
import agent from 'superagent';
import promisifyAgent from 'superagent-promise';
import { JSDOM } from 'jsdom';

const request = promisifyAgent(agent, Bluebird);
const website = 'https://en.wikipedia.org/wiki/List_of_RuPaul%27s_Drag_Race_
↳contestants';
```

Uma informação importante: quando buscamos uma **lista** de elementos do DOM, não nos é retornado um *array* do JavaScript, mas um *NodeList*. Existe uma maneira de contornar isso para trabalhar com os elementos, que é converter a lista de nodos para um *array* nativo. Podemos definir a seguinte função, baseado [neste link](#):

```
function nodeListToArray(dom) {
  return Array.prototype.slice.call(dom, 0);
}
```

Agora, dentro da função principal assíncrona, precisamos extrair os dados. Uma informação importante é que alguns dos itens da lista que precisamos encontram-se no formato `<a href="#">Queen</a>` e outros no formato `Queen`.

```
function main(context, done) {
  // Obter todo o HTML do site em modo texto
  request.get(website).then(({ text }) => {
    // Virtualizar o DOM do texto
    const { window } = new JSDOM(text);
    // Converter os dados da tabela para uma lista e remover os links
    const queens = nodeListToArray(window.document.querySelectorAll('.sortable_
↳tbody tr > td:nth-child(1)'))
      .map(queen => {
        const link = queen.querySelector('a');
        return link === null ? queen.innerHTML : link.innerHTML;
      });

    // Agora, com `queens` contendo a lista que queremos, podemos gerar os alertas
    done({ alerts: queens });
  });
}
```

Basicamente, é possível usar os seletores também utilizados no CSS para extrair os elementos.

4.2 Resolução de problemas

Este documento visa listar alguns problemas comuns durante a criação de uma extensão e como eles podem ser resolvidos.

CHAPTER 5

Ajuda ou parceiras?

Contate-nos em <developer@rung.com.br>