SithasoVant3

Mobile WebApp Development UIKit

Table of contents

1. Introduction

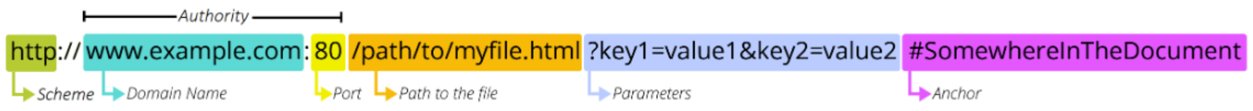
Welcome to SithasoVant3

SithasoVant3 is a tool, a Mobile WebApp UIKit, to help anyone who wants to, to create mobile web based applications (apps). These apps are accessed from the internet. The internet is a collection of computers that are connected together by data streams. Due to the location or placements of these computers, when drawn, they resemble a spider web, thus being called the web.

Creating a web app, involves the process of creating an app that can be accessed anywhere within this web, no matter where one is in the world.

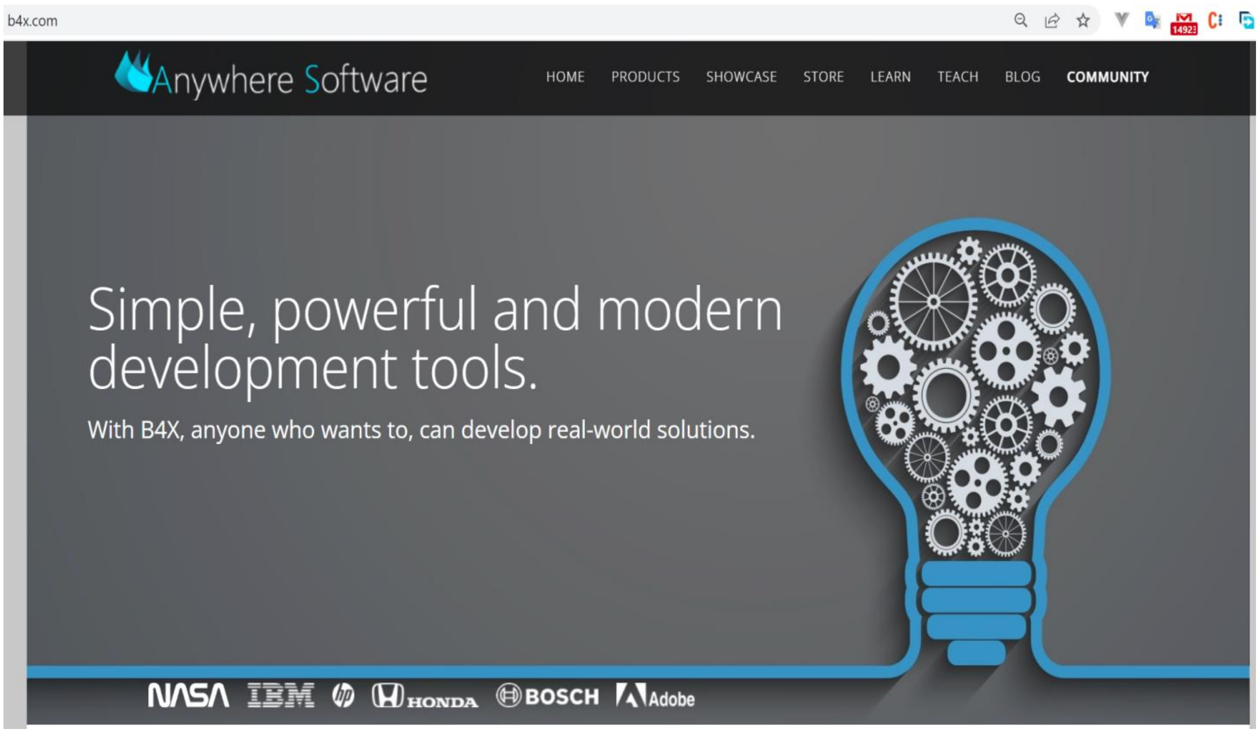
Whilst web apps can be accessed from cellphones and tablets, and even desktops, they are created and then saved on a computer that needs to be online, ie. connected to the web. The computers that host these apps are called web servers. They serve content to the web. People or other resources can then have access to them via a web browser, using a link or URL. There are a variety of these browsers that have been created, for example Google Chrome, Internet Explorer, Mozilla FireFox, Brave etc to browse content on the web. This web is also known as the internet, a global connection of computers that inter-connect. People use this to share content, talk to each other and all other forms of communication like playing games together in different continents etc etc.

To access something on the web, one needs its address, just like a postman needs your residential address to find your place. This address is called a **U**niform **R**esource **L**ocator, i.e. URL. Its friendly known as a **link**. A URL has to be unique, just like your residential address. A typical example of a URL is, https://www.b4x.com/. A URL is composed of different parts, some mandatory and others optional. These are indicated below.


Authority
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
Scheme
Domain Name
Port
Path to the file
Parameters
Anchor

In the URL, the scheme represents the postal service you want to use, the domain name is the city or town, and the port is like the zip code; the path represents the building where your mail should be delivered; the parameters represent extra information such as the number of the apartment in the building; and, finally, the anchor represents the actual person to whom you've addressed your mail.

Just like you write a letter using a language and then post it to be delivered by the postman, the "letters" ie web apps and websites that are created need a specific langauge, a language that an internet browser will understand. This language is made up of 3 parts, these being HTML, CSS and JavaScript. The browser, when you use the URL to find something, finds the webserver hosting the content for that specific address, and then loads the HTML, CSS and JavaScript used on that address and displays a page on the browser. As an example, typing https://www.b4x.com/ on your browser's address bar, displays a nicely formatted page.


b4x.com
Anywhere Software
HOME
PRODUCTS
SHOWCASE
STORE
LEARN
TEACH
BLOG
COMMUNITY
Simple, powerful and modern development tools.
With B4X, anyone who wants to, can develop real-world solutions.

Source: https://www.b4x.com (B4x website), April 15, 2022.

So what is HTML, CSS and JavaScript?

HTML

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.
- HTML files usually have a file extension **.html or .htm**

CSS

- CSS stands for **C**ascading **S**tyle **S**heets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files
- CSS files usually have a file extension **.css**

JAVASCRIPT

- JavaScript is a programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved.

- JavaScript files usually have a file extension **.js**

Creating WebApps with SithasoVant3

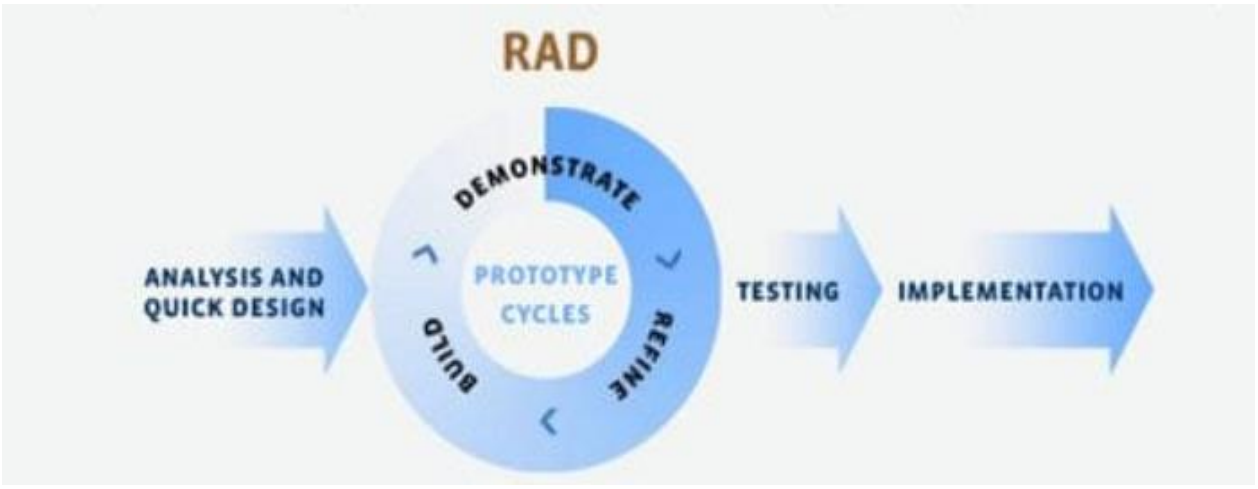
You have an idea and want to share it with the world, why not create a web app for it and other people all over the world can use it? This is what companies like Facebook, Twitter, YouTube and many others have done. When you are creating a web app, you are performing a task known as programming. You need a PC and a programming language. As earlier indicated, the programming languages for the web are HTML, CSS and JavaScript.

You will create web apps with SithasoVant3 using B4X tools. The good thing is, you don't have to know HTML, CSS or JavaScript to do so. SithasoVant3 is also built using B4X. B4X is a Rapid Application Development (RAD) tool, using a cool programming language. The RAD approach is a form of ability to create and respond to change. It is called Agile app development and it prioritizes short times or great rates in development of releases. The RAD model emphasizes the use of software and user feedback over strict planning and requirements recording.

The SithasoVant3 **U**ser **I**nterface **K**it i.e. UIKit, deals with the look and feel i.e. UI, of your application and usually comes with pre-built elements ie the kits, to help one create a web application. The UI is what the person using your web app will see and interact with. This could be buttons, images, videos, music or sound etc. There are 77 UI elements in this SithasoVant3 UI Kit, to help one create powerful and engaging web applications.

When you open TikTok, or Facebook, or Twitter, Instagram, LinkedIn, Telegram etc, all these applications look and feel different when using them, they have different color schemes, animations etc and also provide a nice user interaction as they are simple, engaging and you find yourself using them over and over again. With the SithasoVant3 UIKit, the aim is for you to create such kind of web apps.

Also in RAD, there are 5 steps to be followed during the development of your app, these are...



Step 1: Define and finalize project requirements

During this step, stakeholders sit together to define and finalize project requirements such as project goals, expectations, timelines, and budget. When you have clearly defined and scoped out each aspect of the project's requirements, you can seek management approvals. Stakeholders are all the people who will be involved in developing the app. Questions like how much money is needed to realize their dreams should be answered, what will the app do and achieve, when can it be done by whom should be clearly defined.

Step 2: Begin building prototypes

As soon as you finish scoping the project, you can begin development. Designers and developers will work closely with clients to create and improve upon working draft until the final product is ready.

Step 3: Gather user feedback

In this step, prototypes and beta systems are converted into working models. Developers then gather user feedback to tweak and improve prototypes and create the best possible product.

Step 4: Test, test, test

This step requires you to test your app and ensure that all of its moving parts work together correctly and as the client expects.

Continue adding client feedback while the code is tested and retested to ensure a smooth, functioning final app.

Step 5: Present your system

This is the final step before the finished app goes to launched and available for everyone it needs to reach. It involves data conversion, user training, and (possibly) more testing.

2. Getting Started

The idea you have for an app that can be used by everyone else needs to come to life, however before you can bring it to life, you need to first design it, get it tested, to the limit and get it released.
Your webapp will be made of a single or multiple pages. Each page will follow a particular boiler-plate and will be made of HTML elements. These elements will have a particular look and feel specific to your designs. To be able to make all of this work, you will need to apply attributes and styles to your HTML elements to suit your needs.

From the b4x.com page in the previous section, we noted the color schemes being used, the images, the white and blue text, the dark colors, and all of those followed a pattern.

To understand this pattern, lets please just go through very brief introductions to the building blocks of web apps. These are:

1. Understanding HTML
2. Understanding CSS
3. Understanding JavaScript
4. Understanding Vant
5. Understanding SithasoVant3

In most cases, you will not use this methodology when buildings apps with SithasoVant3, however, there are times when you want to customize something to meet your specific needs, thus the inclusion of this brief introduction.

2.1 Understanding HTML

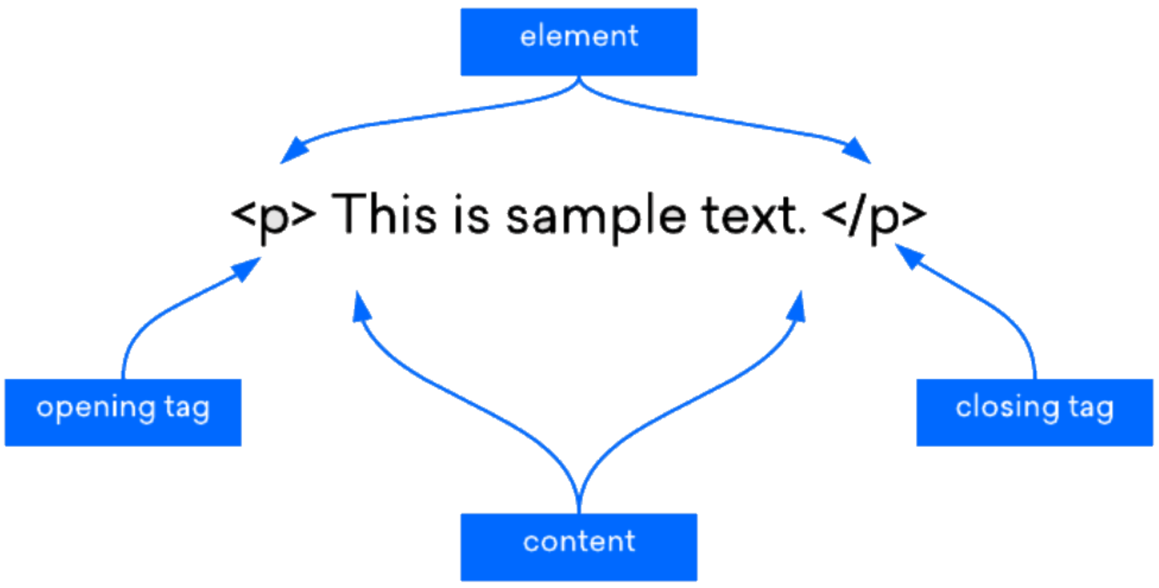
HTML is the language used to create web pages and is used to structure a web page by making use of markup symbols or codes. These are referred to as "elements". These elements are also called “tags”. These tags surround different elements to define the HTML page, and these tell the web browser, like Google Chrome, Safari, Mozilla Firefox, or Brave, how to display the content.

A web page is made up of a collection of these HTML elements.

An HTML tag must contain three parts:

1.An opening tag — this will start with a < > symbol
2. Content — the short instructions on how to display the on-page element
3. A closing tag — this will end with a </ > symbol

This can be demonstrated with



However, some HTML tags can be un-closed. That means that the HTML tag does not need to be closed with a </ >. You'll typically use un-closed tags for metadata or line breaks.

HTML Attributes

A HTML element can have one or more attributes. These provide more information about the HTML and are within the element itself. They can also be used to change color, size, width, height, fonts and other feautures. One common attribute is **style**, which allows you to add style properties to an HTML element. For example

```
<h1 style="font-size:40px;color:green;">This text is 40 pixels and green.</h1>
```

This HTML markup above will be displayed in a web browser as

This text is 40 pixels and green.

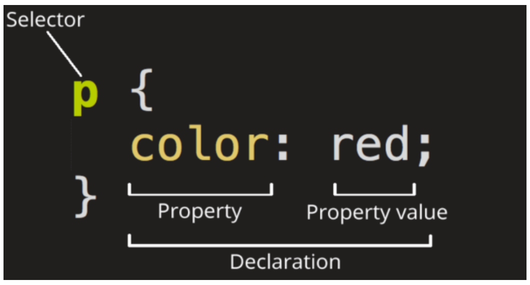
The font size and color applied on this element is done via a method called "**inline-style**", because its part of the text that make the complete HTML element. There is also another method that use the "**class**" attribute. In that case, references to styles used in HTML element is done via a separate **.css** file.

Related Content:
HTML Basics

2.2 Understanding CSS

CSS is a style sheet language that can be used for very basic document text styling — for example, for changing the color and size of headings and links. It can be used to create a layout — for example, turning a single column of text into a layout with a main content area and a sidebar for related information. It can even be used for effects such as animation.

CSS is a **rule-based** language — you define the rules by specifying groups of styles that should be applied to particular elements or groups of elements on your web page. For example, you can decide to have the main heading on your page to be shown as large red text.



```
p {
  color: red;
}
```

Selector

- This is the HTML element name at the start of the ruleset. It defines the element(s) to be styled (in this example, <p> elements). To style a different element, change the selector.

Declaration

- This is a single rule like color: red;. It specifies which of the element's properties you want to style.

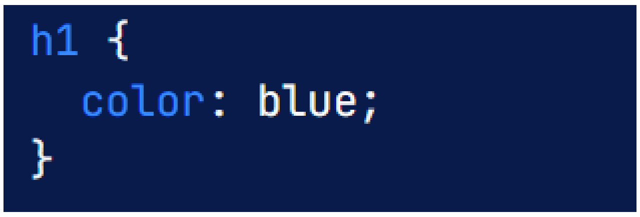
Properties

- These are ways in which you can style an HTML element. (In this example, color is a property of the <p> elements.) In CSS, you choose which properties you want to affect in the rule.

Property value

- To the right of the property—after the colon—there is the property value. This chooses one out of many possible appearances for a given property. (For example, there are many color values in addition to red.)

This example applies a red color to all HTML elements with tag <H1> and also sets the font size.



```
h1 {
  color: blue;
}
```

- In the above example, the CSS rule opens with a **selector** . This selects the HTML element that we are going to style. In this case, we are styling level one headings (<h1>).
- We then have a set of curly braces { }.
- Inside the braces will be one or more **declarations**, which take the form of property and value pairs. We specify the property (color in the above example) before the colon, and we specify the value of the property after the colon (red in this example).
- This example contains two declarations, one for color and the other for font-size. Each pair specifies a property of the element(s) we are selecting (<h1> in this case), then a value that we'd like to give the property.

A CSS stylesheet will contain many such rules, written one after the other. These will be saved in a **.css** file and then added to the HEAD section of your web page, so that they are applied to the referenced HTML elements.

What you saw above were tag css rules, a class rule is written the same way but must have a . infront.

```css
.red-text {
  color: red;
}
```

When defining the HTML element, the class could be added with

```html
<p class="red-text">Here is the first sample of paragraph text.</p>
```

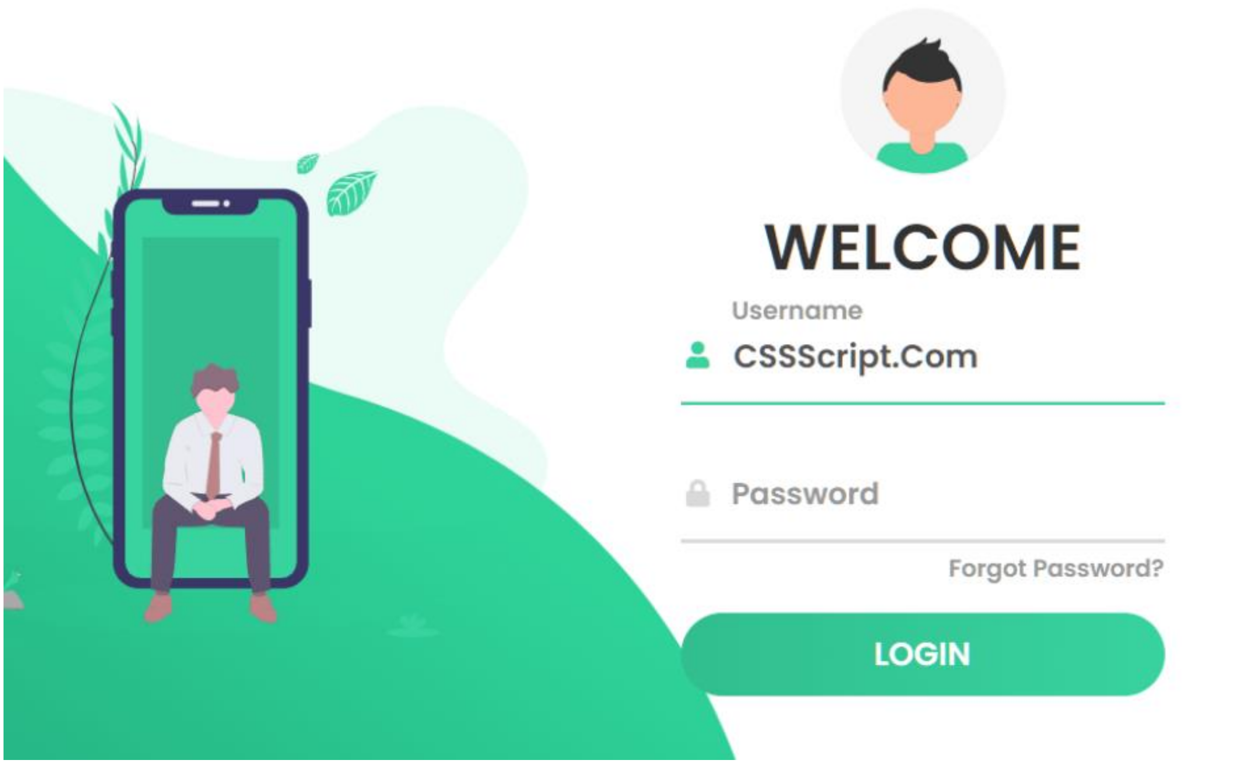
Related Content:
[CSS Basics]

2.3 Understanding JavaScript

JavaScript allows anyone to add complex features to web apps. When you use an application on your mobile phone, the application connects to the Internet and sends data to a webserver. The webserver then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way.

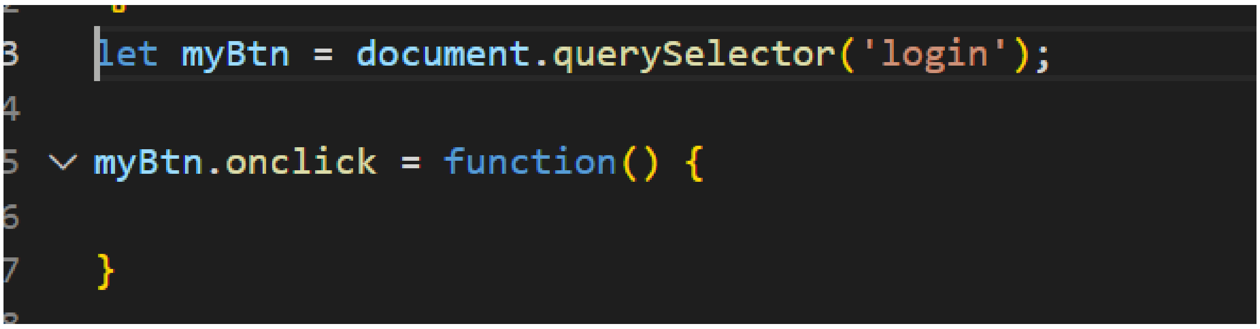
This is what JavaScript does. Lets say you typed in an address, then it asks you to Login, as depicted below. In most cases what you see is a final result based on what the webserver generated and sent to you as a final product. You enter your details and click the "Login" button, Javascript also takes over. For example, lets say you clicke Login without entering anything, through JavaScript, the app should be able to detect through validation and raise an error.

This raising an error is called an "event". Also clicking the login button, fires an event. This event is called "click".



These events, when creating web applications are added with a method called **AddEventListener** to a particular HTML element**.** Some events happen when you move your mouse over elements, when you drag and drop something and all through JavaScript. The event listener will be added to a "click" event for example, and then particular code should be executed to meet what needs to be done.

In this example, it might be, get the entered user details, send them to the server for verification, if valid, show another page or tell end user via a warning that the login credentials are not correct. The javascript code for the button might look like this:



```
let myBtn = document.querySelector('login');

myBtn.onclick = function() {

}
```

Related Content
JavaScript Basics

2.4 Understanding Vant

SithasoVant3 is built on top of Vant. Vant elements are built using the Vue3 web framework. Vue is an approachable, performant and versatile framework for building web user interfaces. These mean the following

Approachable

- Builds on top of standard HTML, CSS and JavaScript with intuitive API (Application Programming Interface) and world-class documentation.

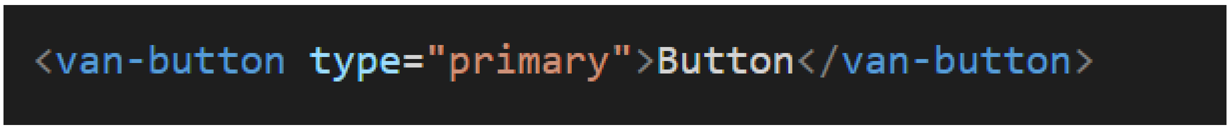
Performant

- Truly reactive, compiler-optimized rendering system that rarely requires manual optimization.

Versatile

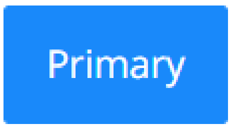
- A rich, incrementally adoptable ecosystem that scales between a library and a full-featured framework.

In the Understanding HTML section, we saw how HTML elements are built using attributes. Following the same pattern, Vant elements use the same methodology. As an example.



```
<van-button type="primary">Button</van-button>
```

This HTML markup will be displayed in the browser as



As you will note here, by just specifying an attribute *type=primary*, the text color of the button is white and its background is blue. That attribute changed the style of the button, just like in the HTML section where the style attribute was used to make the color green. However in this case the style attribute is not written by the developer manually, but done by the Vant framework internally.

Related Content:
Vant

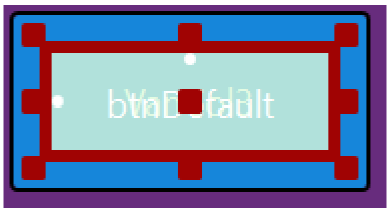
2.5 Understanding SithasoVant3

The SithasoVant3 Mobile WebApp Development Kit, is an attempt to take one out of having to use a text editor to create their web apps, so that one is able to use a performant tool to create their web apps. With SithasoVant3, we talk the b4x tool, the abstract designer, the development IDE (Integrated Development Environment), which all makes it easy to create webapps.

Using the same **van-button** example before, all one has to do is to drop a "view" on an abstract designer/stage, and then set the attribute **type** to **primary**. For SithasoVant3, the HTML element names do not have the "-", so it will be VanButton. One selects it and drops it in an abstract designer, update its properties and then use the generated HTML for that page to create their webapp. To reproduce, one needs to

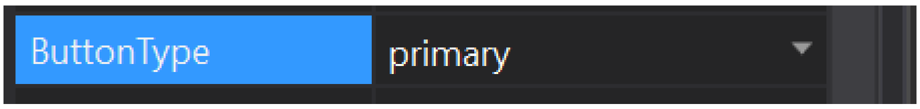
1. Choose VanButton from Custom View Menu

2. Place the generated "view" in its right location



A VanButton (van-button) "view" placed inside another element

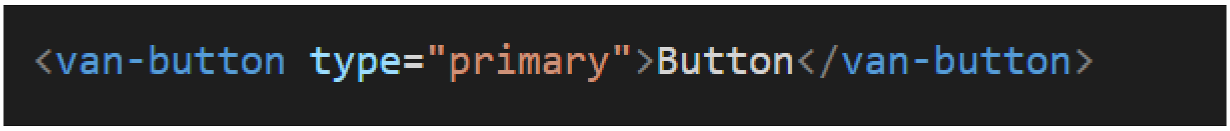
3. Update the needed properties



ButtonType primary

Updating the button "type" to make it primary. We have made this to be a selection for SithasoVant as there are a variety of button types.

When this "view" is loaded, it will automatically create the HTML structure needed to produce this markup



```
<van-button type="primary">Button</van-button>
```

and it also produces the same result as

As we have indicated before, to create a webapp, one can use a collection of HTML elements which do different things. Some are for showing text, playing videos, playing music, linking to other pages etc. The SithasoVant3 framework (based on Vant), has 77 elements which do different things which can be used to create your web apps. These are based on an API. API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. For example by specifying the primary type, internally, the style of the button was changed, by generating the appropriate style attribute of the HTML element.

We can easily call this Vant To HTML, Vant talking to HTML.

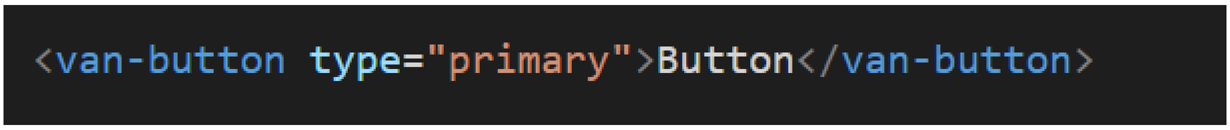
2.5.1 Creating WebApps with SithasoVant3

Before you can create a WebApp with SithasoVant3, you need to plan it and decide which types of Vant elements you will use for it. You can even take a piece of paper and draw on it, how you would like your application to look like. This comes hand in hand with deciding the functionality of your webapp.

To do this, the Vant elements, have been converted to "views" that you can drop on an abstract designer layout and create your user interface. In the B4x language these are called **Custom Views.** The API for each Vant element was converted into custom views that ensures that creating the various elements can be done via the use of a property bag. Below is the van-button API, which indicates the properties that the button has and its options and the default values. Lets take the example of the van-button. Remember this methodology applies to all the elements we will feature here.

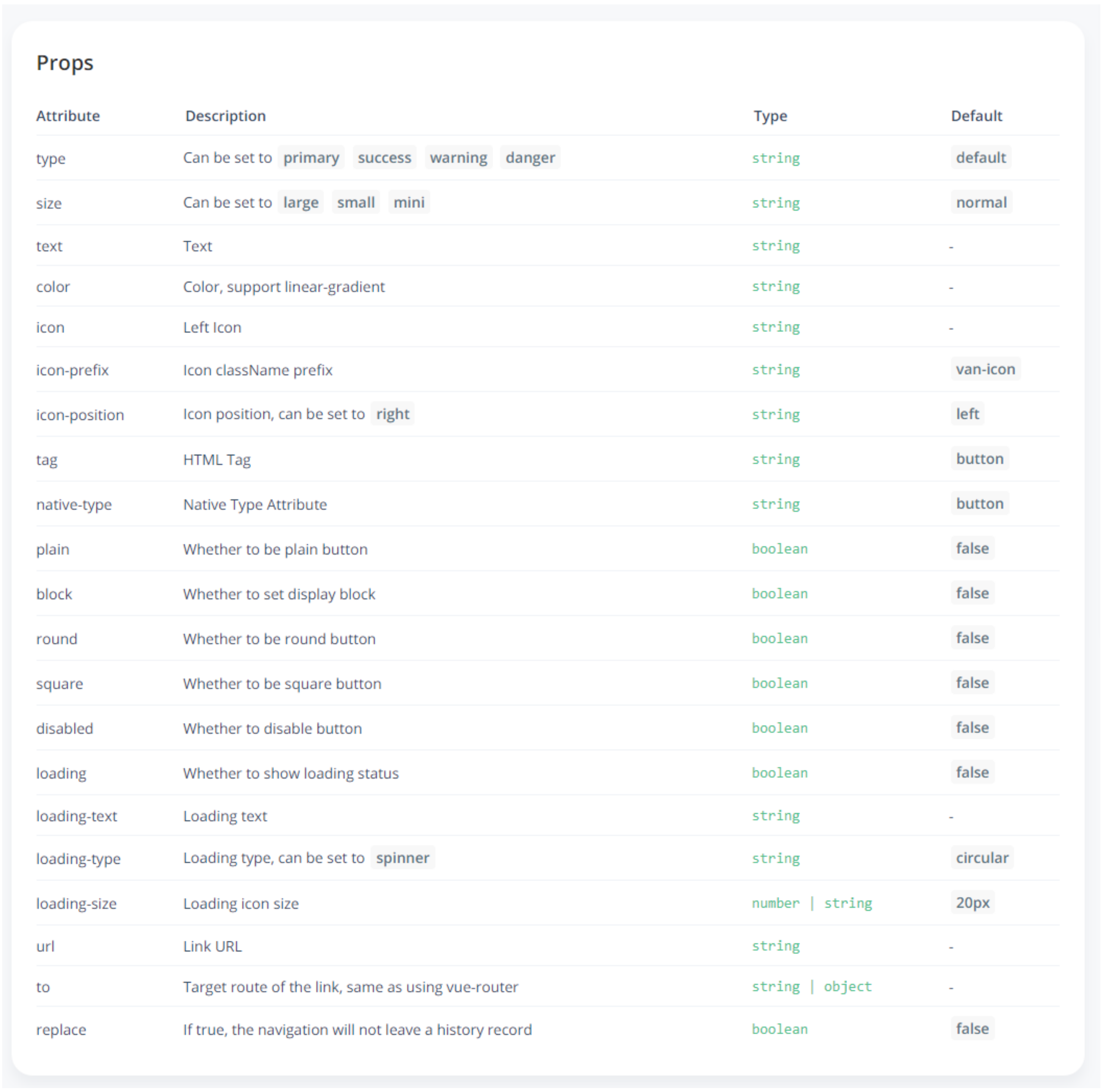
Creating HTML Elements

1. To create a button in Vant, the follow HTML should be defined.



```
<van-button type="primary">Button</van-button>
```

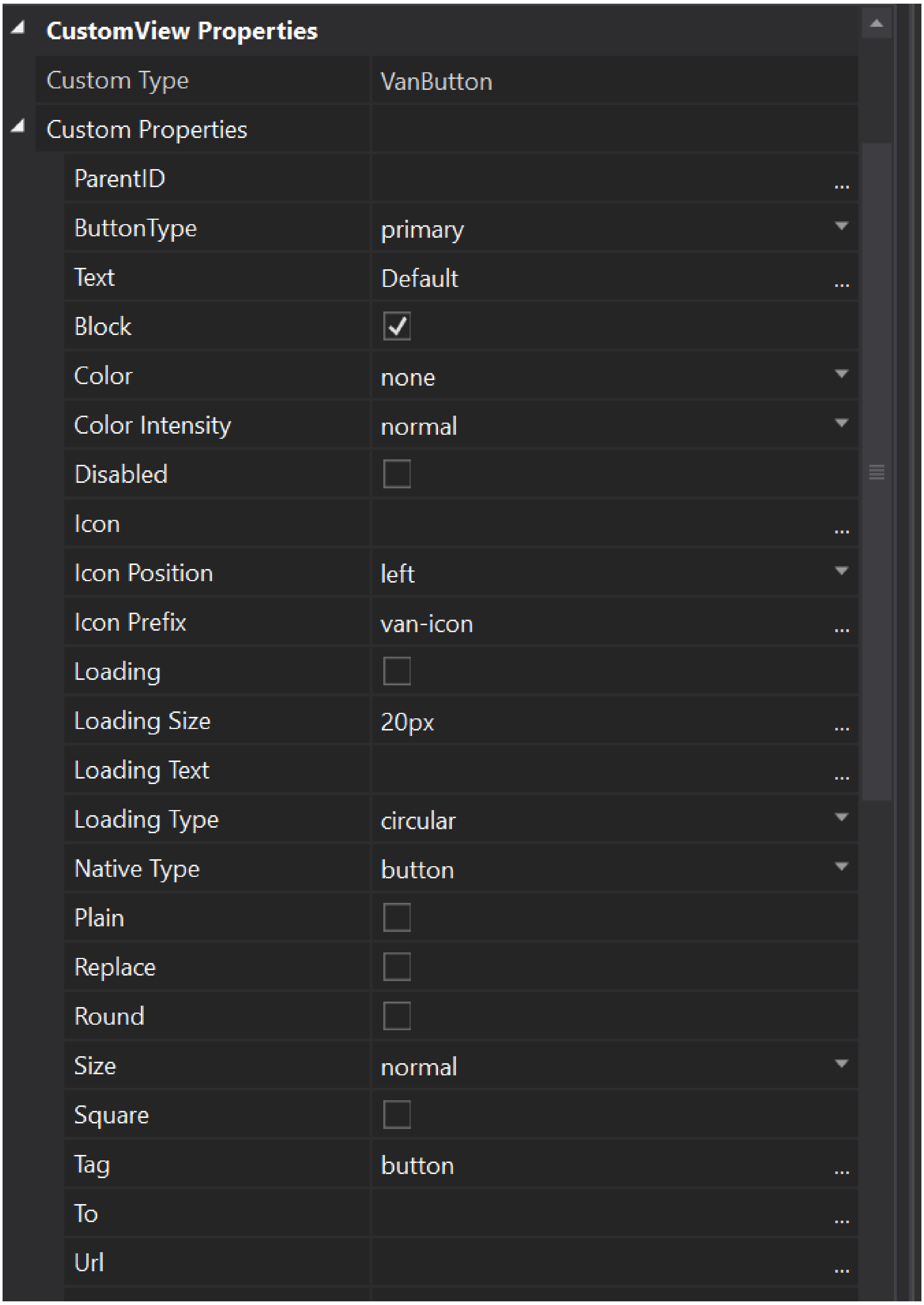
2. The structure of the button can be based on any of the HTML attributes specified on the van-button API.



Props

Attribute	Description	Type	Default
type	Can be set to `primary` `success` `warning` `danger`	string	`default`
size	Can be set to `large` `small` `mini`	string	`normal`
text	Text	string	-
color	Color, support linear-gradient	string	-
icon	Left Icon	string	-
icon-prefix	Icon className prefix	string	`van-icon`
icon-position	Icon position, can be set to `right`	string	`left`
tag	HTML Tag	string	`button`
native-type	Native Type Attribute	string	`button`
plain	Whether to be plain button	boolean	`false`
block	Whether to set display block	boolean	`false`
round	Whether to be round button	boolean	`false`
square	Whether to be square button	boolean	`false`
disabled	Whether to disable button	boolean	`false`
loading	Whether to show loading status	boolean	`false`
loading-text	Loading text	string	-
loading-type	Loading type, can be set to `spinner`	string	`circular`
loading-size	Loading icon size	number | string	`20px`
url	Link URL	string	-
to	Target route of the link, same as using vue-router	string | object	-
replace	If true, the navigation will not leave a history record	boolean	`false`

3. This van-button, for the purposes of SithansoVant3, has been converted to a Property Bag via the Custom View methodology.
This is accessible via the abstract designer in b4x. Where one can specify options via check boxes or select options.

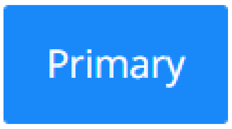


CustomView Properties

Custom Type	VanButton
Custom Properties	
ParentID	...
ButtonType	primary
Text	Default ...
Block	☑
Color	none
Color Intensity	normal
Disabled	☐
Icon	...
Icon Position	left
Icon Prefix	van-icon ...
Loading	☐
Loading Size	20px ...
Loading Text	...
Loading Type	circular
Native Type	button
Plain	☐
Replace	☐
Round	☐
Size	normal
Square	☐
Tag	button ...
To	...
Url	...

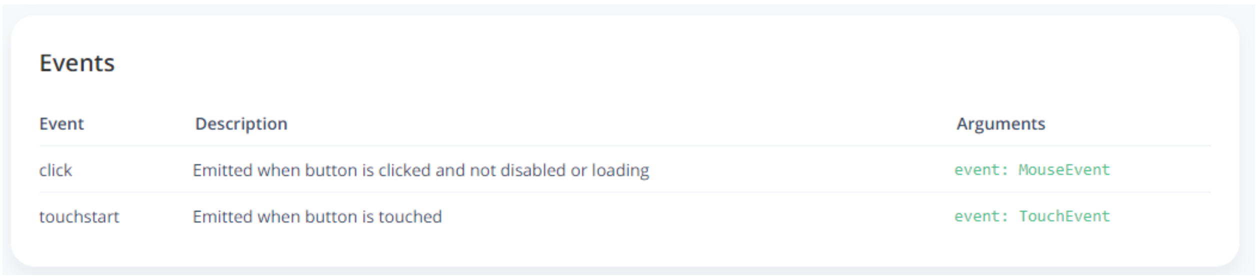
4. For the boolean properties, checkboxes have been used, for those that are optional like the type property,
a choice list has been provided and the rest can just be specified as text. When used, SithasoVant3 internally converts these into usable HTML elements.

Depending on whatever attributes are used from the API in the property bag, a van-button can simply be.



Adding Events

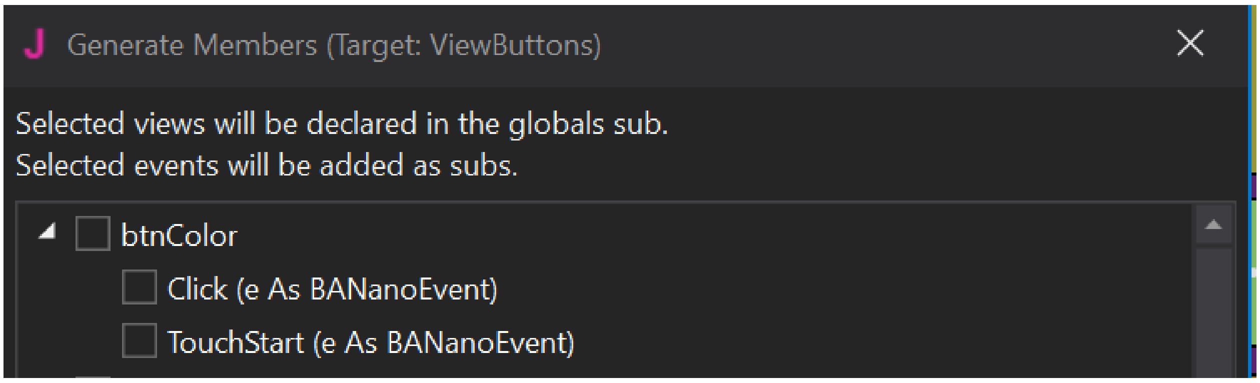
In the Javacript section above we spoke about how "click" events are added to HTML elements. Where available, each element in Vant also has events. The van-button has a "click" and a "touchstart" event as defined in the API. Events are added to elements, after they have been added to a HTML page.



Events

Event	Description	Arguments
click	Emitted when button is clicked and not disabled or loading	`event: MouseEvent`
touchstart	Emitted when button is touched	`event: TouchEvent`

1. To add events using SithasoVant3, in the abstract designer, you click Tools > Generate Members and then select the events you want to add to a HTML element.

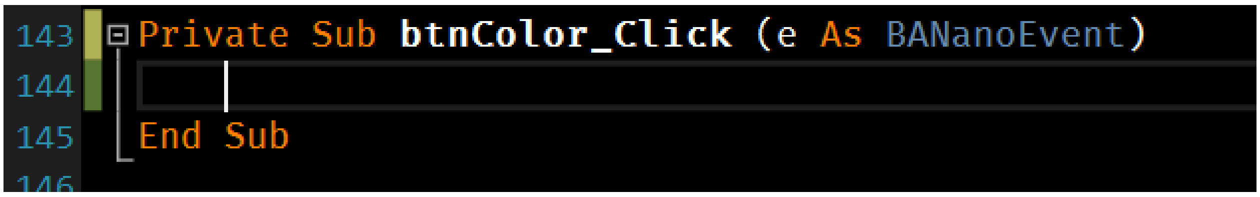


Generate Members (Target: ViewButtons)

Selected views will be declared in the globals sub.
Selected events will be added as subs.

- [ ] btnColor
  - [ ] Click (e As BANanoEvent)
  - [ ] TouchStart (e As BANanoEvent)

2. For our button named **btnColor**, this will produce the following code for us. When a person clicks the button, the code written inside the code block below will be fired.



```
Private Sub btnColor_Click (e As BANanoEvent)

End Sub
```

3. We can then proceed to write code that we need to be executed when that button is clicked by an end user.

We will delve deeper into this topic during the course of this book.

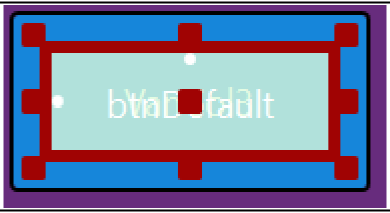
2.5.2 Conventions Used

All the elements in SithasoVant3 have been adopted from the Vant API, however at times how the content is being represented via the B4x Custom Views had to be changed. This ensures that its easy to create webapps via the Property Bag. For each property some extra settings exist.

Things to Know

1. To create the User Interface with SithasoVant3, one uses the Abstract Designer to create the layouts.
2. A parent element is needed for all the elements that should be on a layout. This can be the VanContainer element, which is a simple <div>. All other elements should be put inside it. This is useful for each page you create for your web app.
3. If you put an element inside another in the layout, you are creating a parent child relationship. You can arrange layouts on the tree by dragging and dropping them, this changes their z-order and parent child relationship. To explain this, lets look at the example below.

HTML Code	Output

`<van-cell-group>`
`  <van-cell title="Cell title" value="Content" />`
`</van-cell-group>`
	

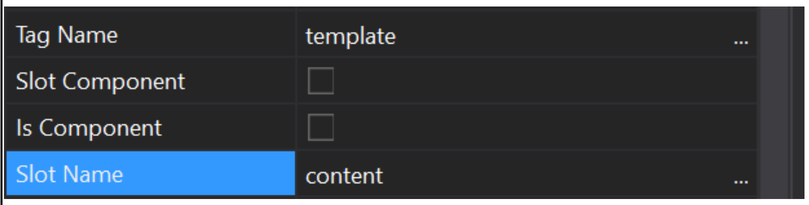
In the example above, in the HTML code, the van-cell is "inside" the van-cell-group. When that is done in the abstract designer, the VanCellGroup should be added first, then followed by the VanCell. Then the VanCell should be dragged and placed inside the VanCellGroup and resized to fit inside it. Starting with the VanCell means you have to fix the zOrder afterwards. So its important to start with adding the parent first and then adding the child items inside the parent element. This also applies to the main parent element for the page.

4. SithasoVant3 is built using Vue3, this adds the possibility to have new attributes like v-html, v-text, v-model, v-show, v-bind (:), v-for, v-if, v-else, v-else-if, v-slot (#), v-on (@). Lets explain some of these directives. The rest are discussed in the additional properties.

Vue Directive	Meaning

: or v-bind	An attribute with a colon infront is known as a binding. Its value will be bound to a data "memory" variable.This can be used to bind the class or style attributes to make them dynamic and be changeable at runtime.
	
	For example to make an image to be changed at runtime, we could bind its src attribute to a memory variable.
	
	`<img v-bind:src="imagesrc" />`
	`'same as`
	`<img :src="imagesrc" />`
	
	`and for changing the image at runtime, call`
	
	`page.SetData("imagesrc", "./assets/image1.png") or later`

Vue Directive	Meaning
	`page.SetData("imagesrc", "./assets/image2.png")`

# or v-slot	Slots act like "optional" html elements inside others. A slot usually has a name. If it is provided, the content will be rendered on the final HTML and if not it wont. For example, lets create a badge on top of a div with 5 	
	
	`<van-badge :content="5">`
	`  <div class="child" />`
	`</van-badge>`
	
	Showing:
	
	Now, instead of showing a number on the badge, lets use a slot to show an icon. So the html element definition will have to be changed.
	
	`<van-badge>`
	`  <div class="child" />`
	`  <template #content>`
	`    <van-icon name="success" class="badge-icon" />`
	`  </template>`
	`</van-badge>`
	
	Meaning that the block "content" should not receive new content, creating
	
	To create a slot in SithasoVant3, you use the VantElement component, update its TagName and SlotName, like this
	
	
	Tag Name: template
	Slot Component: ☐
	Is Component: ☐
	Slot Name: content

@ or v-on	@ or v-on is used to attach an event listener to a HTML element. A typical example would be `<button id="btn1" @:click="doThis"></button> or <button id="btn1" v-on:click="doThis"></button>` **For SithasoVant3 to be able to process the click event linked to this button, 3 things should happen.** **Step 1: Add the subroutine/function via the IDE**

Vue Directive	Meaning

	`Sub doThis`
	`'blah blah blah`
	`End Sub`
	
	Step 2: Add the subroutine/function to Vue / component instance
	
	`btn1.BindState(page)`
	`page.AddMethod("doThis")`
	
	NB:
	
	The .BindState and .AddMethod should be added before the .AddRoute / .Serve call.
	
	For all SithasoVant3 elements, the .AddMethod for all element events will be done automatically after LoadLayout and also .BindState will be executed when .AddRoute/ .Serve is called. This is applicable to own elements defined outside of the provided ones.

5. Besides Vue3 and Vant, SithasoVant3 is powered by B4X and BANano. BANano Transpiles the b4x language to JavaScript, HTML & CSS. The B4x programming langauge uses a similar programming syntax as Visual Basic.

Additional Properties

Property	Definition

Classes	This adds the specified classes to the element. The class names to apply should be separated by a space or ; This has an effect of updating the "class" attribute of the element, which will apply a predefined CSS style to the element.
	
	Values entered here should be linked to some existing CSS rule inside a .css file, for example
	
	`.bold { font-weight: bold; }`
	`.italic { font-style: italic; }`
	
	For example in your element, if you specify a string of "bold italic" in this property, the generated html element for this will be include `class="bold italic"`

Attributes	This adds the specified attributes to the element. The expected format is JSON like, for example 'data-color:red;data-font-size:20px'.
	
	This will ensure that your element has
	
	`data-color="red" data-font-size="20px"`
	
	added as part of the attributes. You can also specify data bound attributes here, for example
	
	`:src=imagesrc`
	
	This will create an attribute on your element like :src="imagesrc",

Property	Definition

	which will expect a data binding, you can either set the value of imagesrc in the States Bindings property or update it with SetData on the Initialize sub routine.
	
	`page.SetData("imagesrc", "./assets/image1.png")`
	
	`If you use the State Bindings property, you dont have to use .SetData to initialize the memory variable.`

v-html	v-html is called a directive in Vue.js and is used to update a element's innerHTML with our data. It accepts a string and render it into HTML. One specifies here the name of a memory "variable", that the content of v-html should be bound to. Changes to the content can be effected by updating the reactive data. If we specify the string txt here...
	
	Example Output would be
	
	`<div v-html="txt"></div>`
	
	`and for changing it call,`
	
	`page.SetData("txt", "<small>This is small text</small>")`

v-model	The v-model directive is used to create two-way data bindings on form input, textarea, and select elements. It binds to the **value** of the element. If you change the input value, the bound data will be changed. One specifies here the name of a memory "variable", that the content of v-model should be bound to. Changes to the content can be affected by updating the reactive data. If we specify the string fullname here...
	
	Example Output would be
	
	`<input v-model="fullname">`
	
	`and for changing it call,`
	
	`page.SetData("fullname", "Anele Mashy Mbanga")`

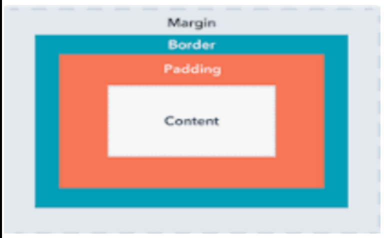
v-show	This is used to toggle an elements visibility. It will also affect the transition if used. One specifies here the name of a memory "variable", that the content of v-show should be bound to. Changes to the content can be affected by updating the reactive data. If we use the string ok here...
	
	Example Output would be (to hide the element by changing the inline style="display:none"
	
	`<h1 v-show="ok">Hello!</h1>`
	
	`and for changing it call,`
	
	`page.SetData("ok", False)`

v-text	v-text works by setting the element's textContent property, so it will overwrite any existing content inside the element. If you need to

Property	Definition

	update the part of textContent, you should use mustache interpolations instead, for example {{ firstname }}. We can use a string msg here or {{ msg }}
	
	Example Output would be
	
	`<span v-text="msg"></span>`
	`<!-- same as -->`
	`<span>{{msg}}</span>`
	
	`and for changing it, call`
	
	`page.SetData("msg", "This is my message to you!")`

TextColor & TextColorIntensity	This updates the style inline color property of your element. A combination of the color and intensity gives out a hex color. For example, selecting color red and intensity lighten-2 will use color "#ef5350", adding the inline style as `style="color:#ef5350"`
BackgroundColor & BackgroundColor Intensity	This updates the style inline background-color property of your element. A combination of the background color and intensity gives out a hex color. For example, selecting color red and intensity lighten-2 will use color "#ef5350", adding the inline style as `style="background-color:#ef5350"`
Border	Checking this property will ensure that the border settings specified are applied to the element based on the specs below.
BorderPosition	Provides functionality to specify where the border should be applied in the element. This can be all, for all corners of the element, top, bottom, left or right. Selecting none does not apply a border. This creates an inline style like style=border-bottom-?, for BorderColor, BorderRadius, BorderStyle and BorderWidth below. Choosing the all position will apply the settings to all border positions.
BorderColor	The color of the border from the provided list of colors. Depending on the position chosen, this creates an inline style like `style=border-top-color:red`
BorderRadius	The radius of the border in px. Depending on the position chosen, this creates an inline style like `style=border-bottom-radius:5px`
BorderStyle	The style of the border. This can be solid, dotted, double, groove etc. Based on the chosen position, this creates a style like `style=border-top-style:solid`
BorderWidth	The width of the border in px. Based on the chosen position, this creates a style like

Property	Definition
	`style=border-right-width:15px`
Margin AXYTBLR 	Margins to be applied to the element using the inline style attribute. A margin is the space around the elements border. So margins are outside the element. For example a=20 will apply a top, bottom, right, left margin of 20px to the element x=20 will apply a left and right margin of 20px to the element y=20 will apply a top and bottom margin of 20px to the element t=20 will apply a top margin of 20px to the element b=20 will apply a bottom margin of 20px to the element l=20 will apply a left margin of 20px to the element r=20 will apply a right margin of 20px to the element This creates an inline style like `style=margin-top:20px; margin-left:20px`
Padding AXYTBLR	Padding to be applied to the element using the inline style attribute. Padding is the space between the elements border and the elements content. So padding is inside the element. For example a=20 will apply a top, bottom, right, left padding of 20px to the element x=20 will apply a left and right padding of 20px to the element y=20 will apply a top and bottom padding of 20px to the element t=20 will apply a top padding of 20px to the element b=20 will apply a bottom padding of 20px to the element l=20 will apply a left padding of 20px to the element r=20 will apply a right padding of 20px to the element This creates an inline style like `style=padding-top:20px; padding-left:20px`
Height	The style height to be applied to the element in px. A value of 20px here will update the inline style to 20px e.g. `style=height:20px;`
Width	The style width to be applied to the element in px. A value of 20px here will update the inline style to 20px e.g. `style=width:20px;`

v-for	This directive is used to render the element or template block multiple times based on the source data. The directive's value must use the special syntax **alias in expression** to provide an alias for the current element being iterated on. For example
	
	`<div v-for="item in items">`
	`  {{ item.text }}`
	`</div>`

key	The key property is usually used with the v-for directive to ensure that. It is special because, its primarily used as a hint for Vue's virtual DOM algorithm to identify vnodes when diffing the new list of nodes against the old list. It is important to always use it for v-for usage. For example, the example above could have been written as `<div v-for="item in items" :key="item.id">`
v-if	The v-if directive is used to conditionally render an element or a

Property	Definition
	template fragment based on the truthy-ness of the expression value. One specifies here the name of a memory "variable", that the content of v-model should be bound to. Changes to the content can be affected by updating the reactive data. If we specify a string awesome here... The block will only be rendered if the directive's expression returns a truthy value. If false, the block will NOT exist in the DOM. Example Output could be `<h1 v-if="awesome">Vue is awesome!</h1>` **`and for adding the block to the DOM, call`** `page.SetData("awesome", True)`
v-else	You can use the v-else directive to indicate an "else block" for v-if: Example Output would be `<button @click="awesome = !awesome">Toggle</button>` `<h1 v-if="awesome">Vue is awesome!</h1>` `<h1 v-else>Oh no 😢</h1>`
v-else-if	The v-else-if, as the name suggests, serves as an "else if block" for v-if. It can also be chained multiple times: Example output would be `<div v-if="type === 'A'">A</div>` `<div v-else-if="type === 'B'">B</div>` `<div v-else-if="type === 'C'">C</div>` `<div v-else>Not A/B/C</div>` **`and for changing it, call`** **`page.SetData("type", "A")`**
State Bindings	Data bound attributes should be initialized to be able for them to be reactive, ie, updatable at runtime. The State Bindings property is to help do that. Here one specifies all the data bound attributes initial values. Anything entered here is only useful in the Initialize sub routine. For example, entering **`imagesrc=./assets/image1.png`** **`will fire`** **`page.SetData("imagesrc", "./assets/image1.png")`** **`after LoadLayout before .Serve / .AddRoute`** **`You can also apply other types of bindings here, for example, assuming x as a data-binding variable, writing...`**

Property	Definition

	`x=boolean - calls page.SetData("x", false)`
	`x=true - calls page.SetData("x", true)`
	`x=false - calls page.SetData("x", false)`
	`x=array or x=list - calls page.SetData("x", page.NewList). NewList creates a new list.`
	`x=object or x=map - calls page.SetData("x", page.NewMap). NewMap creates a new map.`
	`x=string - calls page.SetData("x", "")`
	`x=int - calls page.SetData("x", 0)`
	`x=null - calls page.SetData("x", Null)`

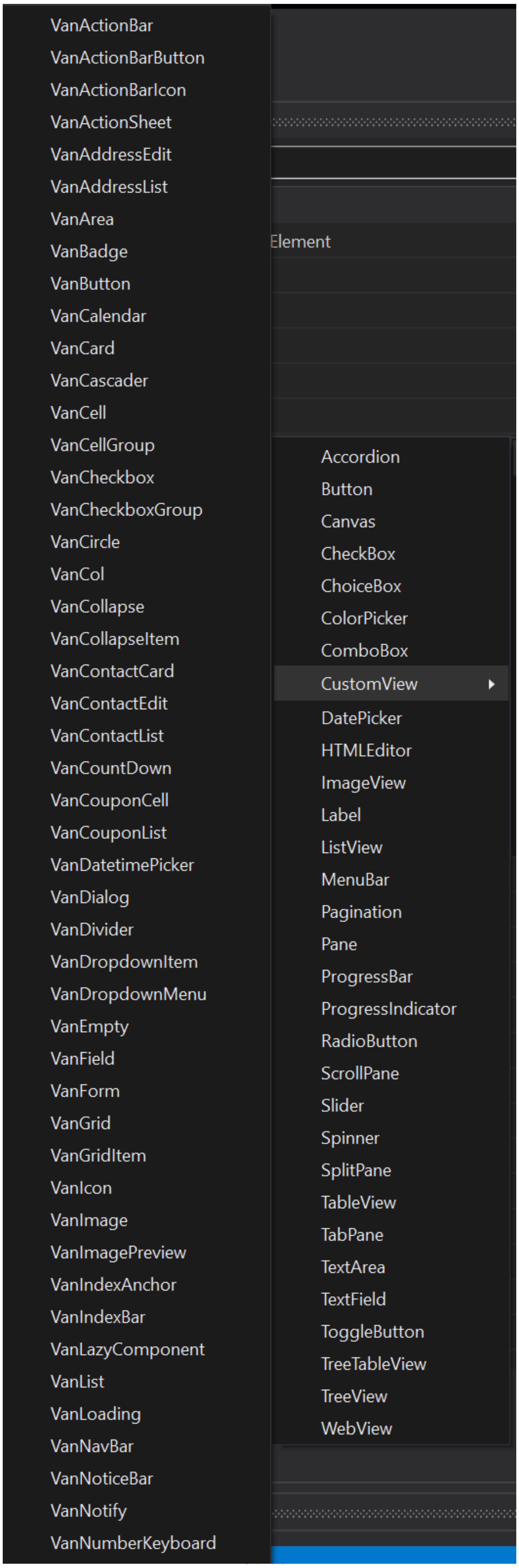
3. Mobile WebApp UI Building Blocks

To build webapps with SithasoVant3, you can choose between the following types of HTML elements:

Component Types	**Available Components**
Basic Components	Button, Cell, Icon, Image, Layout, Popup, Toast
Form Components	Calendar, Cascader, CheckBox, DatetimePicker, Field, Form, NumberKeyboard, PasswordInput, Picker, Radio, Rate, Search, Slider, Stepper, Switch, Uploader
Action Components	ActiionSheet, Dialog, DropdownMenu, Loading, Notify, Overlay, PullRefresh, ShareSheet, SwipeCell
Display Components	Badge, Circle, Collapse, CountDown, Divider, Empty, ImagePreview, LazyLoad, List, NoticeBar, Popover, Progress, Skeleton, Steps, Sticky, Swipe, Tag
Navigation Components	ActionBar, Grid, IndexBar, NavBar, Pagination, Sidebar, Tab, Tabbar, TreeSelect
Business Components	AddressEdit, AddressList, Area, Card, ContactCard, ContactEdit, ContactList, Coupon, SubmitBar

You are also not limited to these elements and can use any HTML element of your choice like H1, p, span, etc.

As indicated before, creating these via SithasoVant3 is based on using the abstract designer and also generate the respective events needed to make your webapp execute what is needed.


VanActionBar
VanActionBarButton
VanActionBarIcon
VanActionSheet
VanAddressEdit
VanAddressList
VanArea
VanBadge
VanButton
VanCalendar
VanCard
VanCascader
VanCell
VanCellGroup
VanCheckbox
VanCheckboxGroup
VanCircle
VanCol
VanCollapse
VanCollapseItem
VanContactCard
VanContactEdit
VanContactList
VanCountDown
VanCouponCell
VanCouponList
VanDatetimePicker
VanDialog
VanDivider
VanDropdownItem
VanDropdownMenu
VanEmpty
VanField
VanForm
VanGrid
VanGridItem
VanIcon
VanImage
VanImagePreview
VanIndexAnchor
VanIndexBar
VanLazyComponent
VanList
VanLoading
VanNavBar
VanNoticeBar
VanNotify
VanNumberKeyboard
Element
Accordion
Button
Canvas
CheckBox
ChoiceBox
ColorPicker
ComboBox
CustomView
DatePicker
HTMLEditor
ImageView
Label
ListView
MenuBar
Pagination
Pane
ProgressBar
ProgressIndicator
RadioButton
ScrollPane
Slider
Spinner
SplitPane
TableView
TabPane
TextArea
TextField
ToggleButton
TreeTableView
TreeView
WebView

The Vant elements here, as noted are based on HTML elements and by changing their attributes anyone can change them to suit their needs. For each of the elements below, we will showcase its HTML structure and the resulting output on a mobile device. This is just to give you a background how each element is made and how it will look like as the final output.

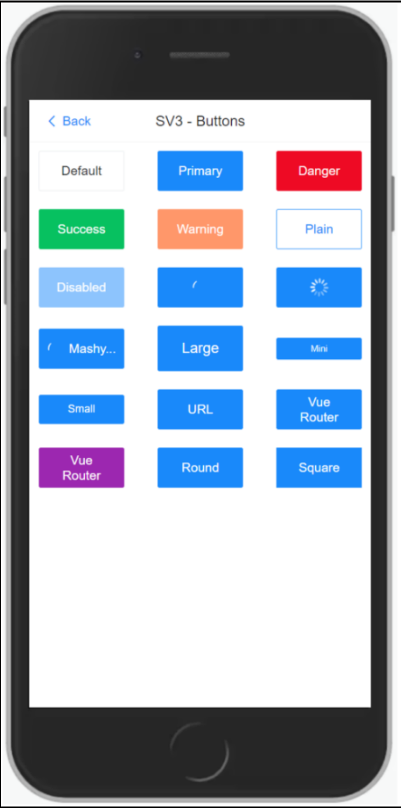
As you will notice, some elements need extra settings to make them functional. Some of these settings can only be effected in code and or via the property bag. We will discuss how to update settings via code in the next chapters.

3.1 Basic Components

3.1.1 Button

Buttons are used to trigger an action, such as submitting a form.

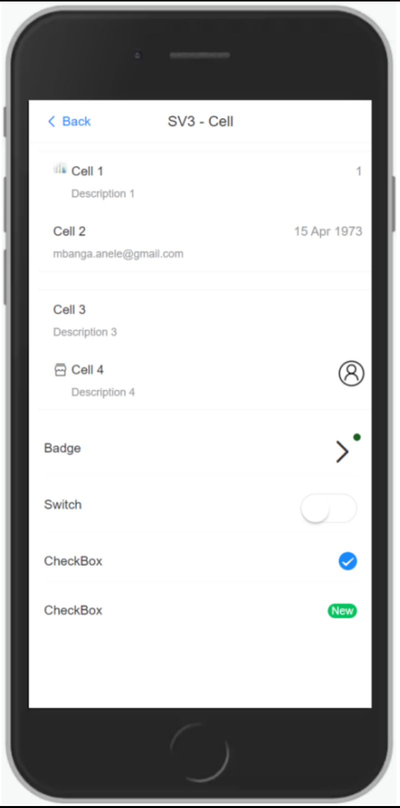
HTML	Output

`<van-button type="primary">Primary</van-button>`
`<van-button type="success">Success</van-button>`
`<van-button type="default">Default</van-button>`
`<van-button type="danger">Danger</van-button>`
`<van-button type="warning">Warning</van-button>`
	 Back
	SV3 - Buttons
	Default
	Primary
	Danger
	Success
	Warning
	Plain
	Disabled
	Mashy...
	Large
	Mini
	Small
	URL
	Vue Router
	Vue Router
	Round
	Square

3.1.2 Cell

The cell is a single display item in the list.

HTML	Output

`<van-cell-group>`
`  <van-cell title="Cell title" value="Content" />`
`  <van-cell title="Cell title" value="Content"`
`label="Description" />`
`</van-cell-group>`
	

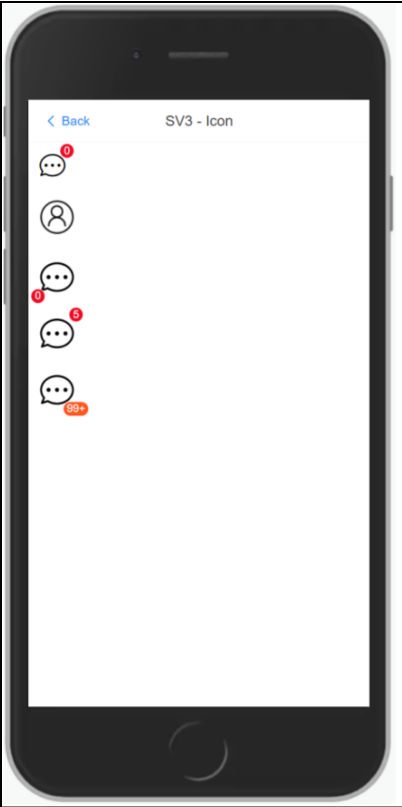
< Back SV3 - Cell

Cell 1 Description 1	1
Cell 2 mbanga.anele@gmail.com	15 Apr 1973
Cell 3 Description 3	
Cell 4 Description 4	
Badge	
Switch	
CheckBox	
CheckBox	New

3.1.3 Icon

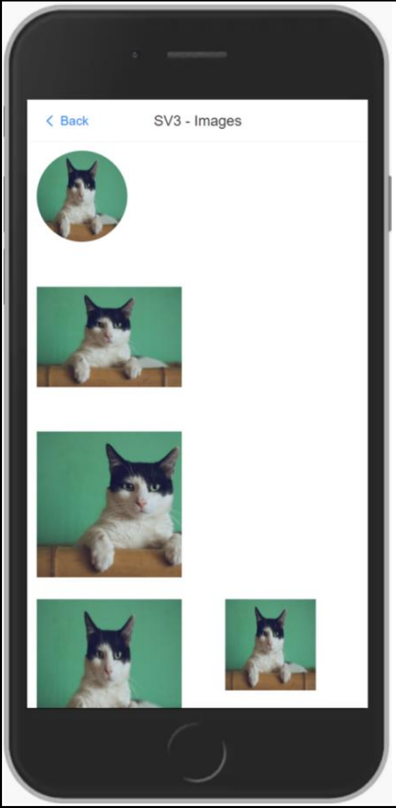
The font-based icon set that can be used via the Icon component or referenced in other components via the `icon` attribute.

HTML	Output

`<van-icon name="chat-o" dot />`
`<van-icon name="chat-o" badge="9" />`
`<van-icon name="chat-o" badge="99+" />`
	

3.1.4 Image

Enhanced img tag with multiple image fill modes, support for image lazy loading, loading hint, loading failure hint.

HTML	Output
`<van-image width="100" height="100" src="https://cdn.jsdelivr.net/npm/@vant/assets/cat.jpeg" />`	

3.1.5 Layout

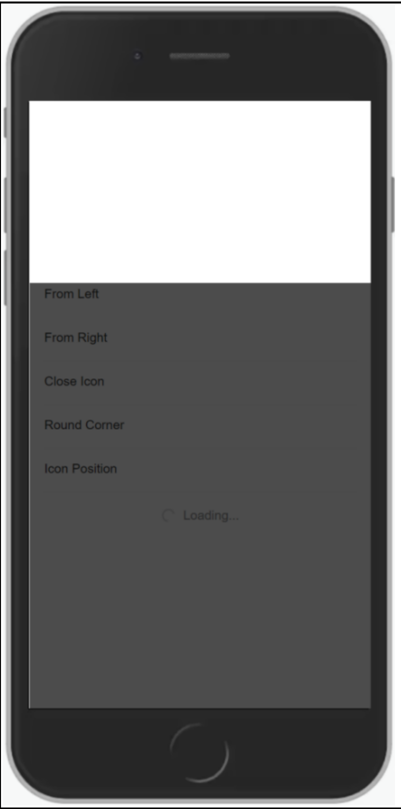
Quickly and easily create layouts with `van-row` and `van-col`.

HTML	Output

```
<van-row>
  <van-col span="8">span: 8</van-col>
  <van-col span="8">span: 8</van-col>
  <van-col span="8">span: 8</van-col>
</van-row>

<van-row>
  <van-col span="4">span: 4</van-col>
  <van-col span="10" offset="4">offset: 4,
span: 10</van-col>
  <van-col span="6">span: 6</van-col>
</van-row>

<van-row>
  <van-col offset="12" span="12">offset:
12, span: 12</van-col>
</van-row>
```

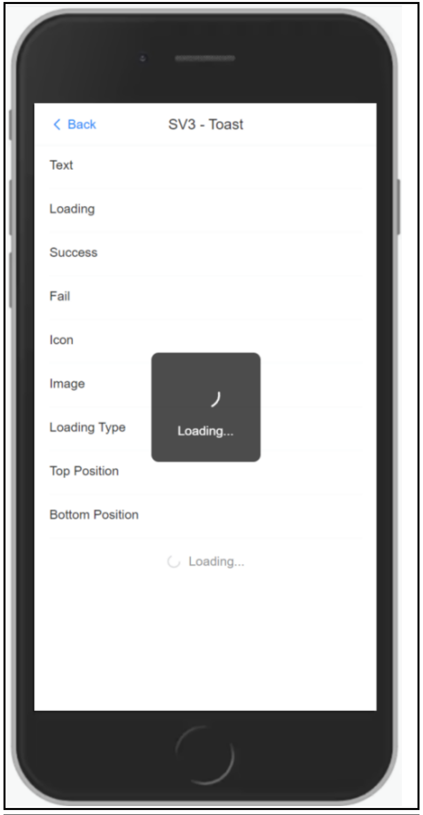
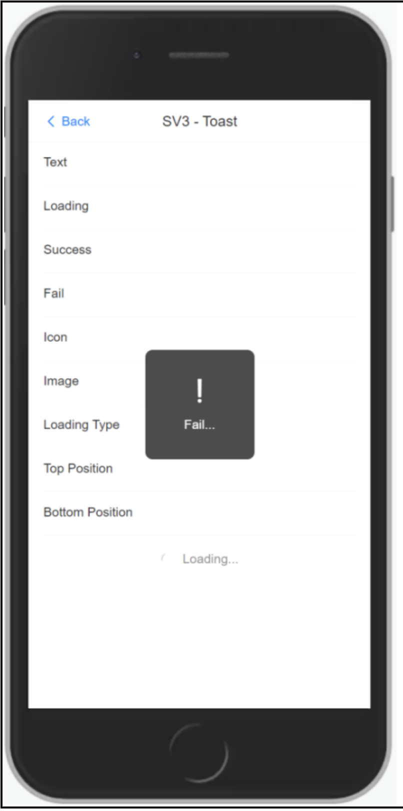
3.1.6 Popup

Used to display pop-up windows, information prompts, etc., and supports multiple pop-up layers to display.

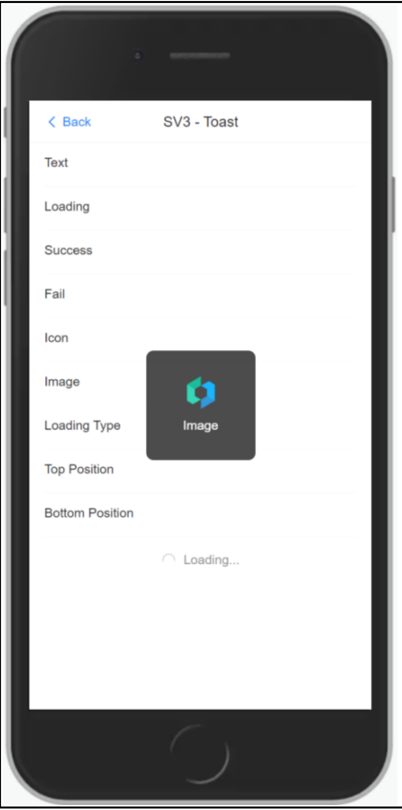
HTML	Output
`<van-popup v-model:show="show" position="top" :style="{ height: '30%' }" />`	 From Left From Right Close Icon Round Corner Icon Position Loading...

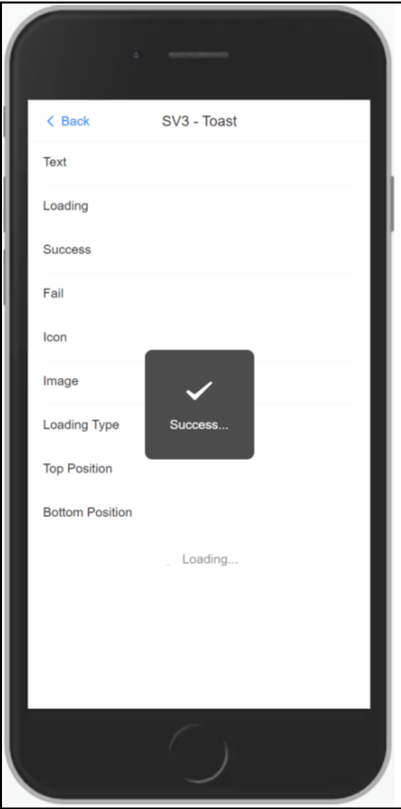
3.1.7 Toast

Black semi-transparent pop-up hint in the middle of the page, used for message notification, loading hint, operation result hint and other scenarios.

B4X Code	Output
`myToast.Defaults.ShowToastLoading(vapp, "Loading...")`	 Back SV3 - Toast Text Loading Success Fail Icon Image Loading Type Top Position Bottom Position Loading... Loading...
`myToast.Defaults.ShowToastFail(vapp, "Fail...")`	 Back SV3 - Toast Text Loading Success Fail Icon Image Loading Type Top Position Bottom Position ! Fail... Loading...

B4X Code	Output

`myToast.Icon = "https://cdn.jsdelivr.net/npm/@vant/assets/logo.png"`
`myToast.ShowToast(vapp, "Image")`
	 Back SV3 - Toast Text Loading Success Fail Icon Image Loading Type Top Position Bottom Position Image Loading...

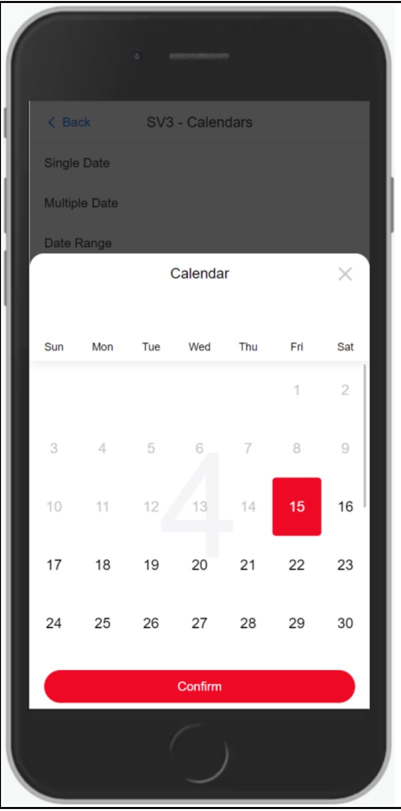
`myToast.Defaults.ShowToastSuccess(vapp, "Success...")`	 Back SV3 - Toast Text Loading Success Fail Icon Image Loading Type Top Position Bottom Position Success... Loading...

3.2 Form Components

3.2.1 Calendar

Calendar component for selecting dates or date ranges.

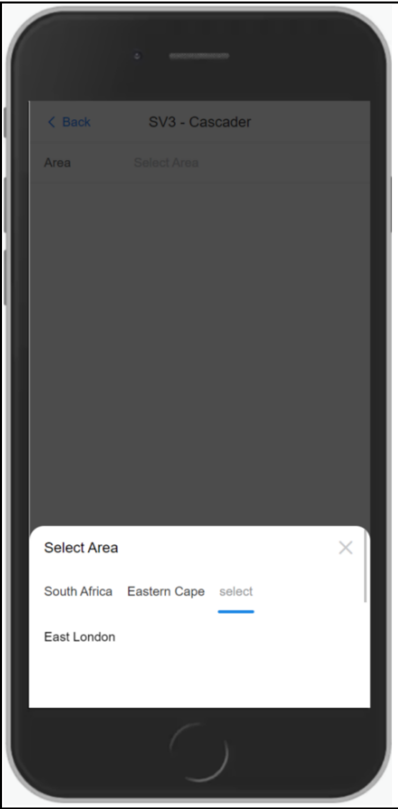
HTML Code	Output

`<van-cell title="Select Single Date" :value="date" @click="show = true" />`
`<van-calendar v-model:show="show" @confirm="onConfirm" />`
	 Back
	SV3 - Calendars
	Single Date
	Multiple Date
	Date Range
	Calendar
	Sun Mon Tue Wed Thu Fri Sat
	15
	Confirm

3.2.2 Cascader

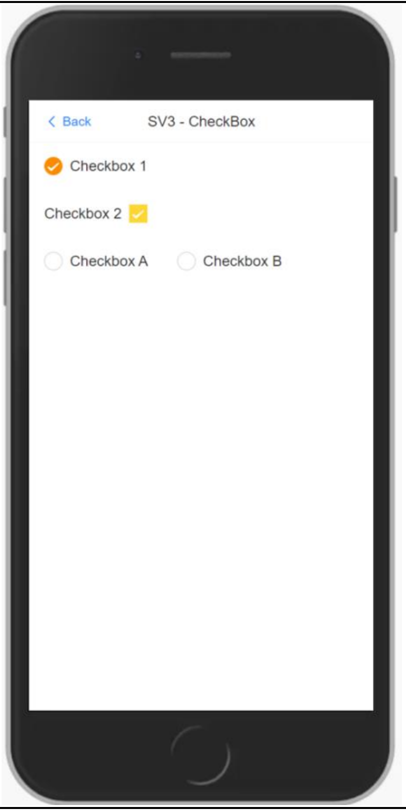
The cascader component is used for the selection of multi-level data. The typical scene is the selection of provinces and cities.

HTML Code	Output

`<van-field`
`  v-model="fieldValue"`
`  is-link`
`  readonly`
`  label="Area"`
`  placeholder="Select Area"`
`  @click="show = true"`
`/>`
`<van-popup v-model="show" round position="bottom">`
`  <van-cascader`
`    v-model="cascaderValue"`
`    title="Select Area"`
`    :options="options"`
`    @close="show = false"`
`    @finish="onFinish"`
`  />`
`</van-popup>`
	

3.2.3 Checkbox

A group of options for multiple choices.

HTML	Output
`<van-checkbox v-model="checked">Checkbox</van-checkbox>`	

Back

SV3 - CheckBox

Checkbox 1

Checkbox 2

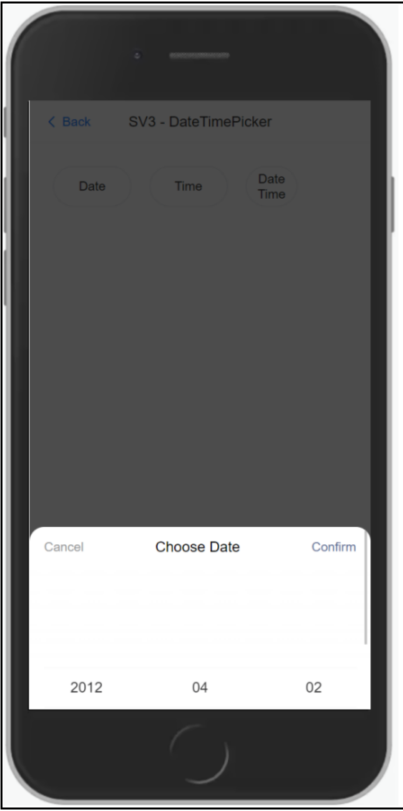
Checkbox A

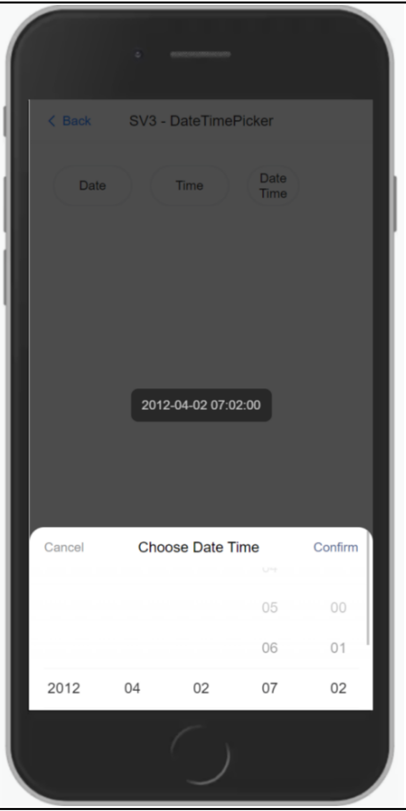
Checkbox B

3.2.4 DatetimePicker

Used to select time, support date and time dimensions, usually used with the Popup component.

HTML Code	Output

```
<van-datetime-picker
  v-model="currentDate"
  type="date"
  title="Choose Date"
  :min-date="minDate"
  :max-date="maxDate"
/>
```
	 Back
	SV3 - DateTimePicker
	Date
	Time
	Date Time
	Cancel
	Choose Date
	Confirm
	2012
	04
	02

```
<van-datetime-picker
  v-model="currentTime"
  type="time"
  title="Choose Time"
  :min-hour="10"
  :max-hour="20"
/>
```
	 Back
	SV3 - DateTimePicker
	Date
	Time
	Date Time
	2012-04-02 07:02:00
	Cancel
	Choose Date Time
	Confirm
	05
	00
	06
	01
	2012
	04
	02
	07
	02

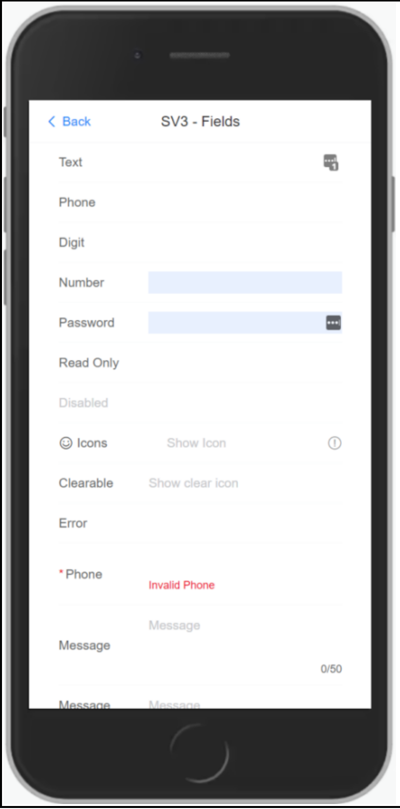
3.2.5 Field

Field component let users enter and edit text.

HTML Code

```
<van-cell-group inset>
  <van-field v-model="text" label="Text" />
  <van-field v-model="tel" type="tel" label="Phone" />
  <van-field v-model="digit" type="digit" label="Digit"
/>
  <van-field v-model="number" type="number"
label="Number" />
  <van-field v-model="password" type="password"
label="Password" />
</van-cell-group>
```

Output

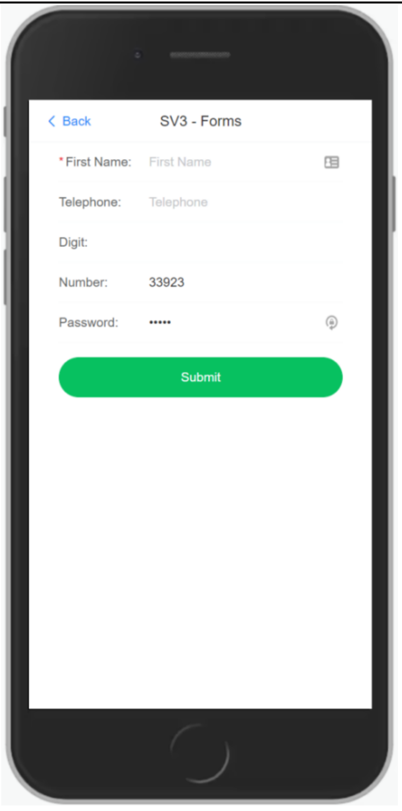


3.2.6 Form

Used for data entry and verification, and supports input boxes, radio buttons, check boxes, file uploads and other types. Should be used with Field component.

HTML Code	Output

```html
<van-form @submit="onSubmit">
  <van-cell-group inset>
    <van-field
      v-model="username"
      name="Username"
      label="Username"
      placeholder="Username"
      :rules="[{ required: true, message: 'Username is required' }]"
    />
    <van-field
      v-model="password"
      type="password"
      name="Password"
      label="Password"
      placeholder="Password"
      :rules="[{ required: true, message: 'Password is required' }]"
    />
  </van-cell-group>
  <div style="margin: 16px;">
    <van-button round block type="primary" native-type="submit">
      Submit
    </van-button>
  </div>
</van-form>
```



< Back SV3 - Forms

* First Name:	First Name
Telephone:	Telephone
Digit:	
Number:	33923
Password:	•••••

3.2.7 NumberKeyboard

The NumberKeyboard component can be used with PasswordInput component or custom input box components.

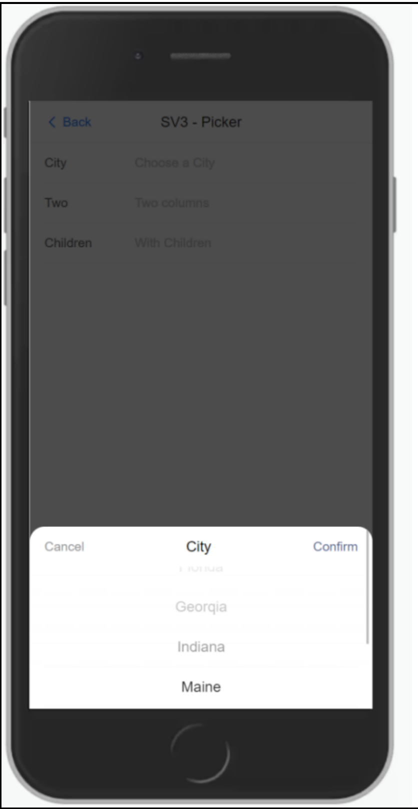
3.2.8 PasswordInput

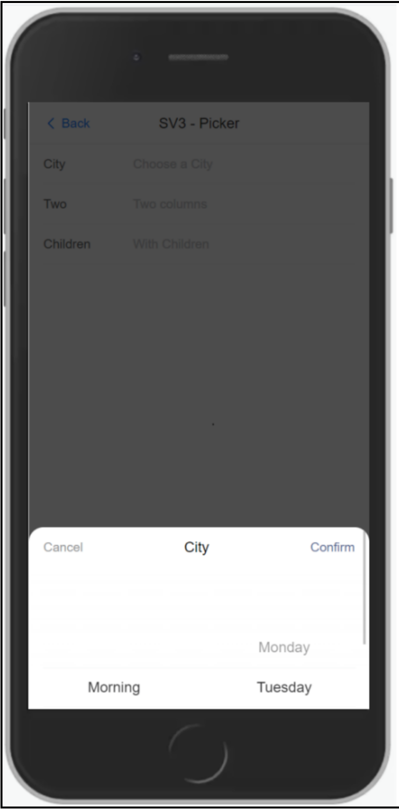
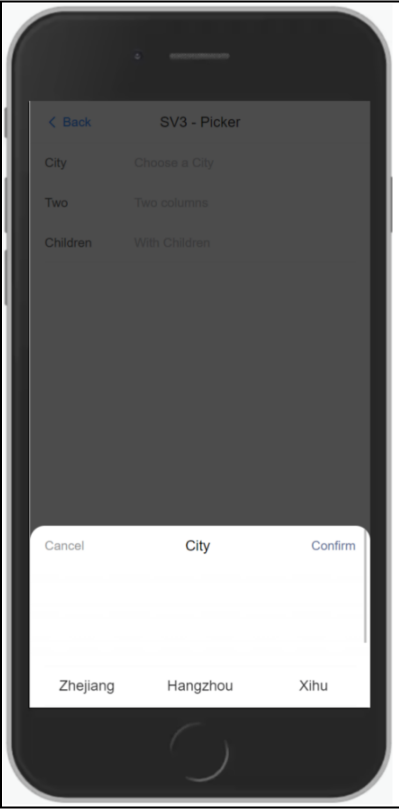
The PasswordInput component is usually used with NumberKeyboard Component.

3.2.9 Picker

The picker component is usually used with Popup Component.

HTML Code	Output

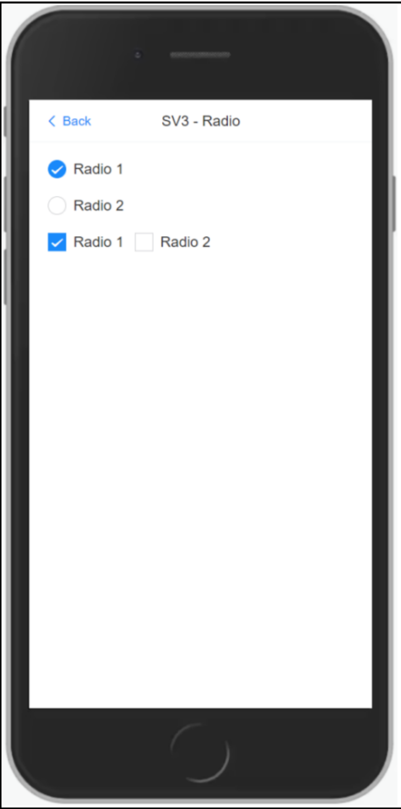
`<van-picker`
`  title="Title"`
`  :columns="columns"`
`  @confirm="onConfirm"`
`  @cancel="onCancel"`
`  @change="onChange"`
`/>`
	 Back
	SV3 - Picker
	City
	Choose a City
	Two
	Two columns
	Children
	With Children
	Cancel
	City
	Confirm
	Georgia
	Indiana
	Maine

HTML Code	Output
`<van-picker ref="picker" title="Title" :columns="columns" @change="onChange" />`	 Back SV3 - Picker City Choose a City Two Two columns Children With Children Cancel City Confirm Monday Morning Tuesday
	 Back SV3 - Picker City Choose a City Two Two columns Children With Children Cancel City Confirm Zhejiang Hangzhou Xihu

3.2.10 Radio

Single selection among multiple options.

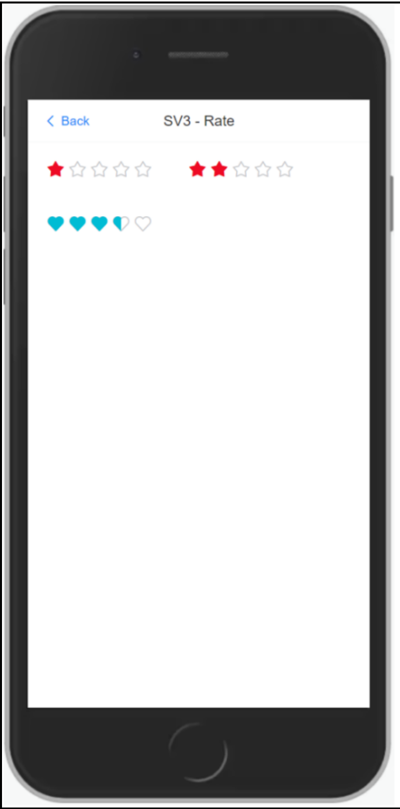
HTML Code	Output

`<van-radio-group v-model="checked">`
`  <van-radio name="1">Radio 1</van-radio>`
`  <van-radio name="2">Radio 2</van-radio>`
`</van-radio-group>`
	 Back
	SV3 - Radio
	Radio 1
	Radio 2
	Radio 1
	Radio 2

3.2.11 Rate

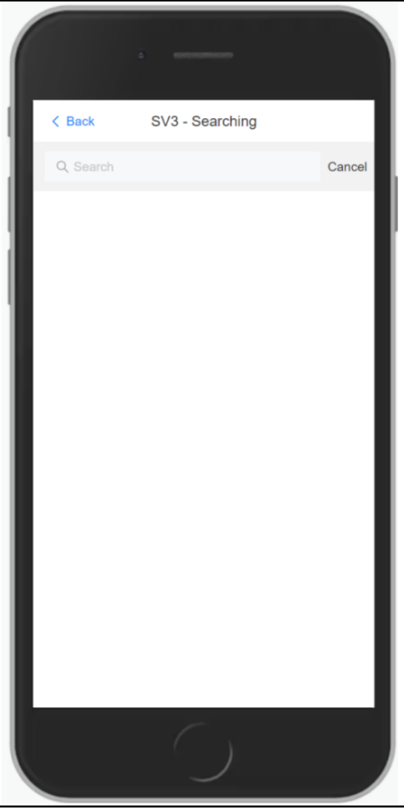
The rate component is used for rating things.

HTML Code	Output

```
<van-rate
  v-model="value"
  :size="25"
  color="#ffd21e"
  void-icon="star"
  void-color="#eee"
/>
```
	

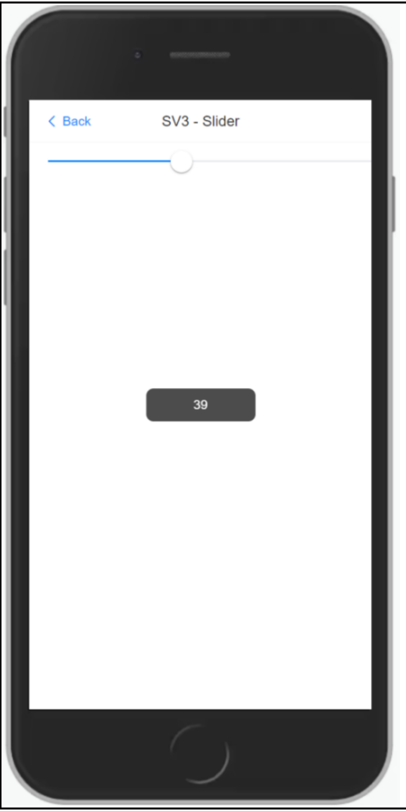
3.2.12 Search

Input box component for search scenarios.

HTML Code	Output
`<van-search v-model="value" placeholder="Placeholder" />`	 Back SV3 - Searching Search Cancel

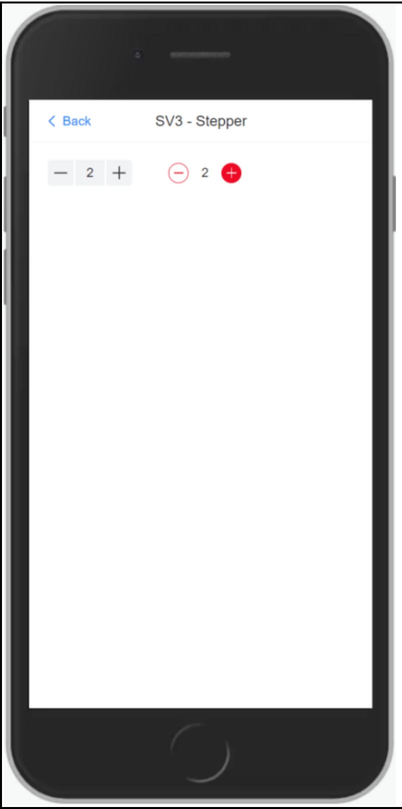
3.2.13 Slider

Used to select a value within a given range.

HTML Code	Output
`<van-slider v-model="value" @change="onChange" />`	

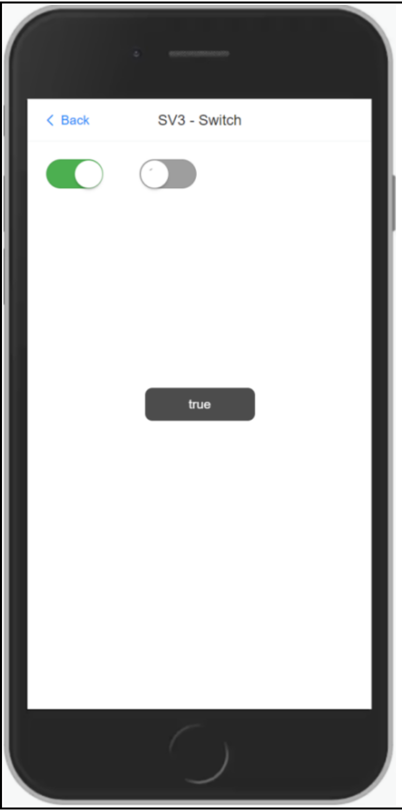
3.2.14 Stepper

The stepper component consists of an increase button, a decrease button and an input box, which are used to stepper and adjust numbers within a certain range.

HTML Code	Output
`<van-stepper v-model="value" min="5" max="8" />`	

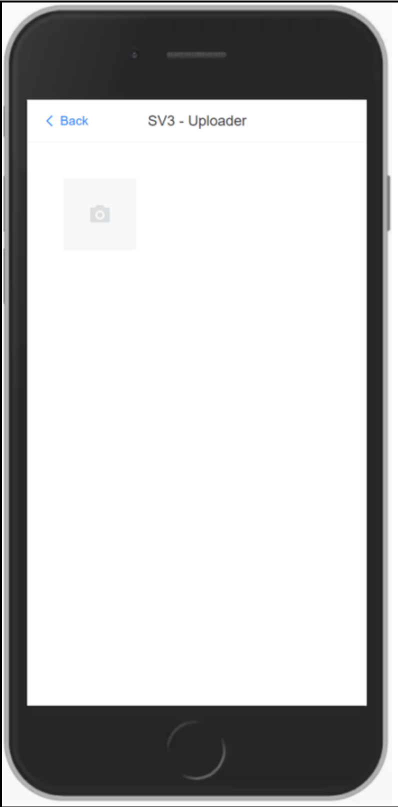
3.2.15 Switch

Used to switch between open and closed states.

HTML Code	Output
`<van-switch v-model="checked" />`	

3.2.16 Uploader

Used to upload a local image or file to the server and display a preview image and upload progress during the upload process. The Uploader component does not currently contain the interface logic for uploading files to the server, this step needs to be implemented by the user.

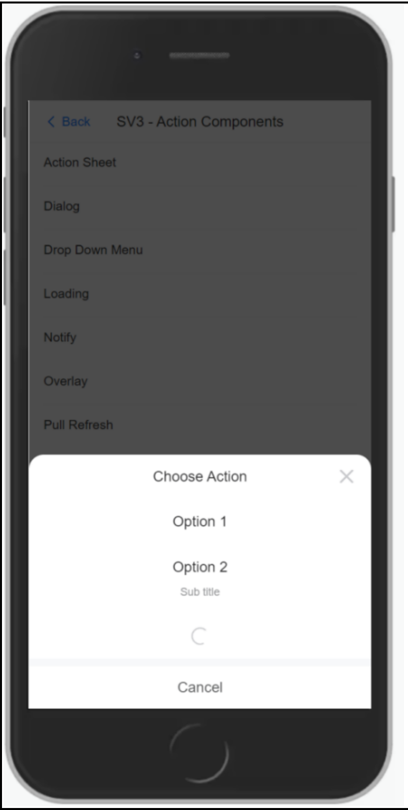
HTML Code	Output
`<van-uploader v-model="fileList" :after-read="afterRead" />`	

3.3 Action Components

3.3.1 ActionSheet

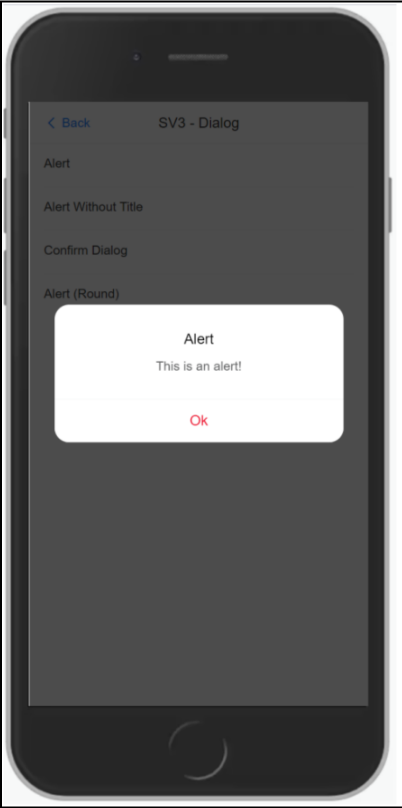
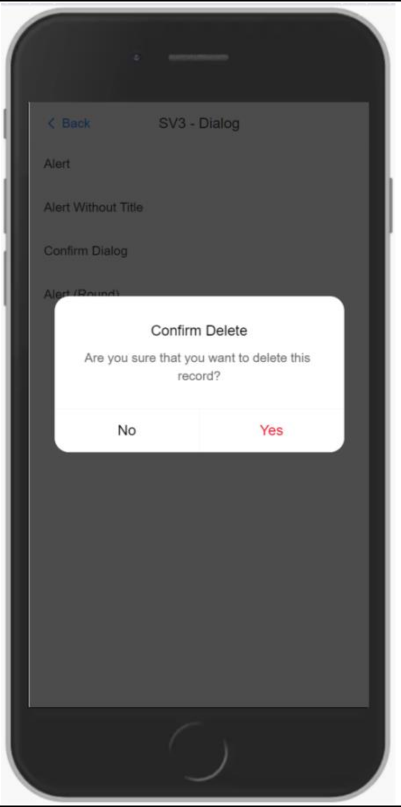
The pop-up modal panel at the bottom contains multiple options related to the current situation.

HTML Code	Output

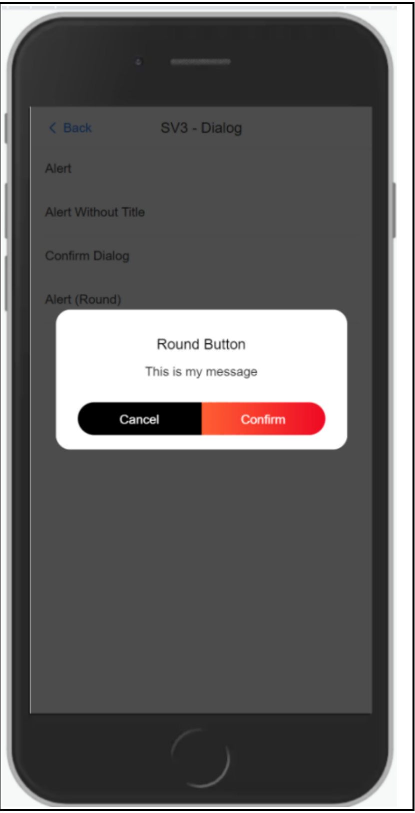
`<van-cell is-link title="Basic Usage" @click="show = true" />`
`<van-action-sheet v-model:show="show" :actions="actions" @select="onSelect" />`
	

3.3.2 Dialog

A modal box pops up on the page, which is often used for message prompts, message confirmation, or to complete specific interactive operations in the current page. It supports two methods: function call and component call.

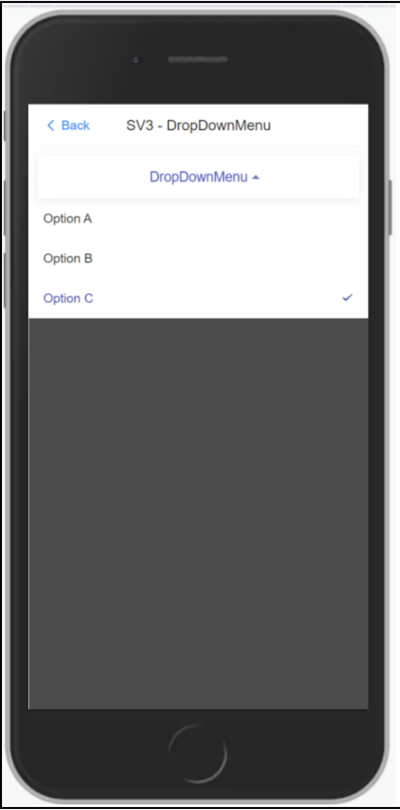
B4X Code	Output
`VanDialog1.ShowAlert("alert", "Alert", "This is an alert!", "Ok")`	 Back SV3 - Dialog Alert Alert Without Title Confirm Dialog Alert (Round) Alert This is an alert! Ok
`VanDialog1.ShowConfirm("delete", "Confirm Delete", "Are you sure that you want to delete this record?", "Yes", "No")`	 Back SV3 - Dialog Alert Alert Without Title Confirm Dialog Confirm Delete Are you sure that you want to delete this record? No Yes

```
VanDialog1.Defaults
VanDialog1.Theme = VanDialog1.THEME_ROUND_BUTTON
VanDialog1.Title = "Round Button"
VanDialog1.Message = "This is my message"
VanDialog1.Result = "rounder"
VanDialog1.Show
```



3.3.3 DropdownMenu

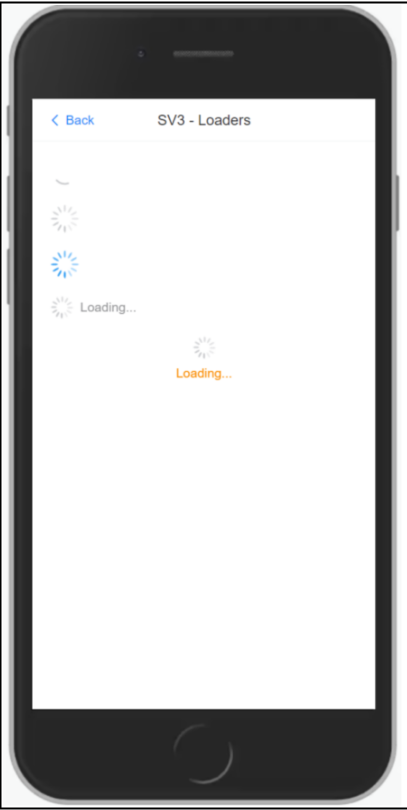
The menu list that pops down downwards.

HTML Code	Output
`<van-dropdown-menu>` `<van-dropdown-item v-model="value1" :options="option1"` `/>` `<van-dropdown-item v-model="value2" :options="option2"` `/>` `</van-dropdown-menu>`	

3.3.4 Loading

Used to indicate the transition state during loading.

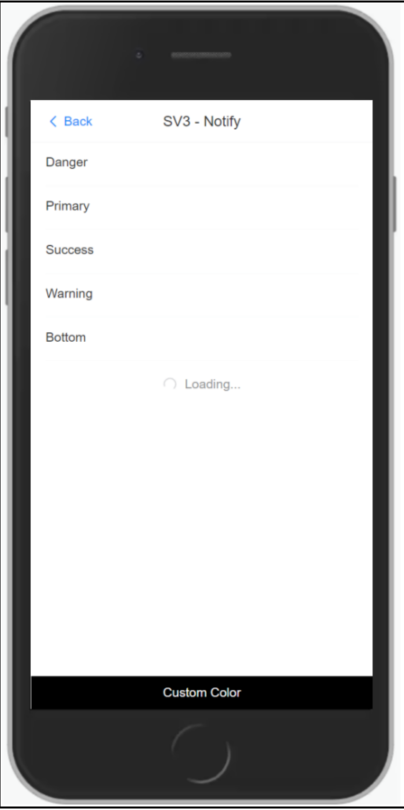
HTML Code	Output

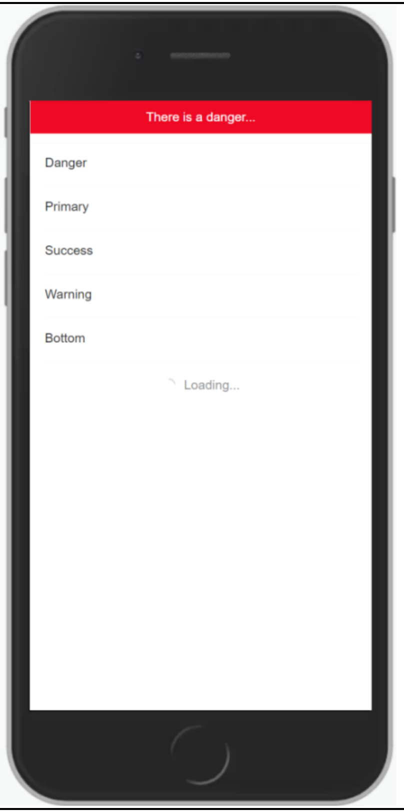
`<van-loading />`
`<van-loading type="spinner" />`
`<van-loading size="24px" vertical>Loading...</van-loading>`
	

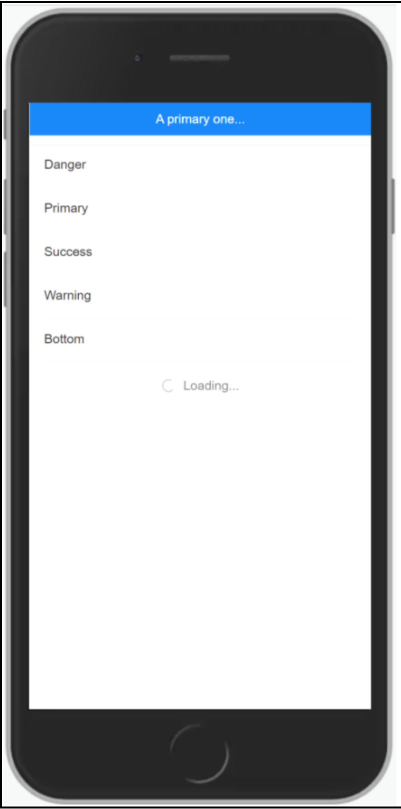
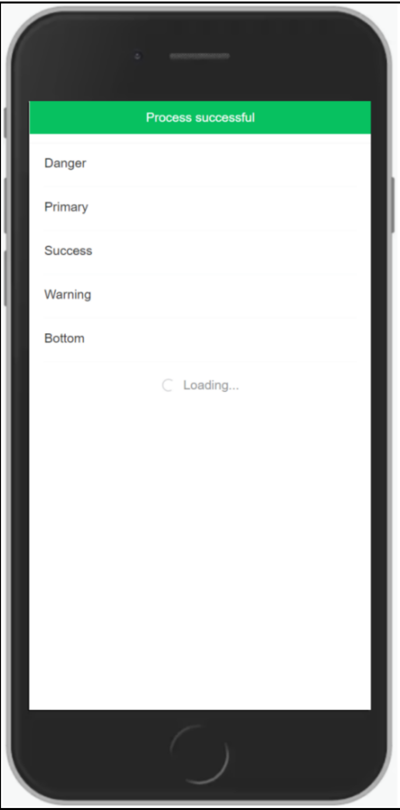
3.3.5 Notify

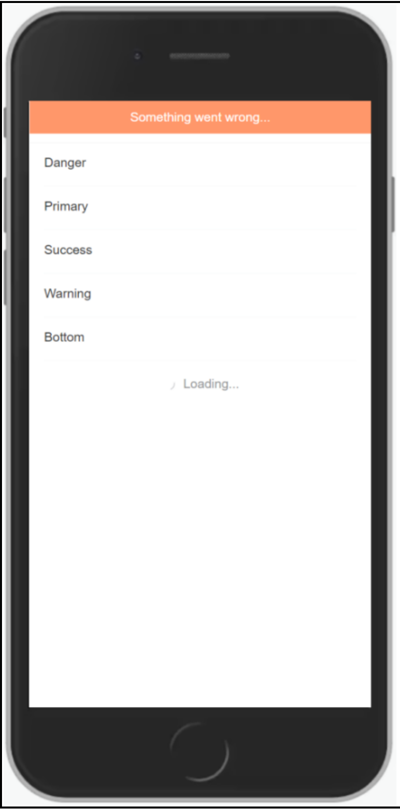
The display message prompt is at the top of the page, and supports two methods: function call and component call.

B4X Code	Output

`VanNotify1.Position = VanNotify1.NOTIFY_POSITION_BOTTOM`
`VanNotify1.Color = BANanoShared.COLOR_WHITE`
`VanNotify1.Background = BANanoShared.COLOR_BLACK`
`VanNotify1.Message = "Custom Color"`
`VanNotify1.Show`
	 Back
	SV3 - Notify
	Danger
	Primary
	Success
	Warning
	Bottom
	Loading...
	Custom Color

`VanNotify1.ShowDanger("There is a danger...")`	 There is a danger... Danger Primary Success Warning Bottom Loading...

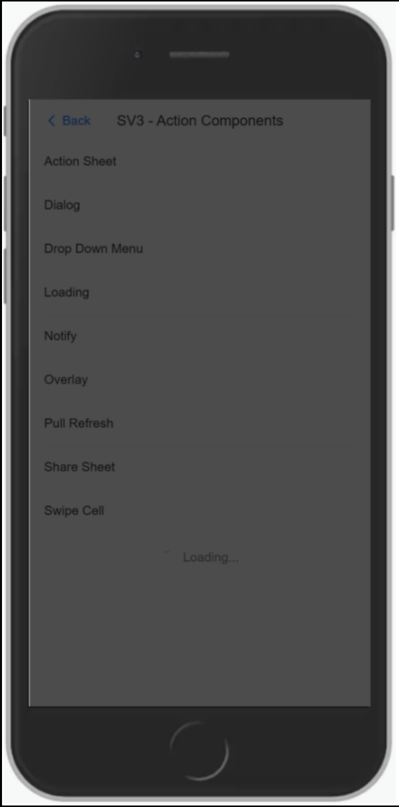
B4X Code	Output
`VanNotify1.ShowPrimary("A primary one...")`	 A primary one... Danger Primary Success Warning Bottom Loading...
`VanNotify1.ShowSuccess("Process successful")`	 Process successful Danger Primary Success Warning Bottom Loading...

B4X Code	Output
`VanNotify1.ShowWarning("Something went wrong...")`	 Something went wrong... Danger Primary Success Warning Bottom Loading...

3.3.6 Overlay

Create a mask layer to emphasize specific page elements and prevent users from performing other operations.

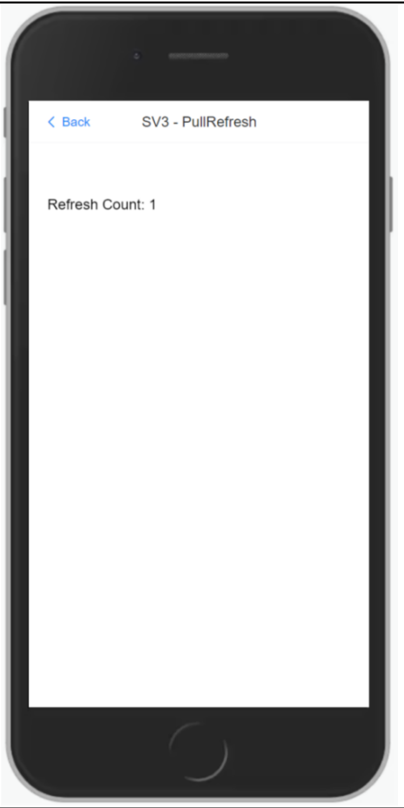
HTML Code	Output

`<van-button type="primary" text="Show Overlay" @click="show = true" />`
`<van-overlay :show="show" @click="show = false" />`
	

3.3.7 PullRefresh

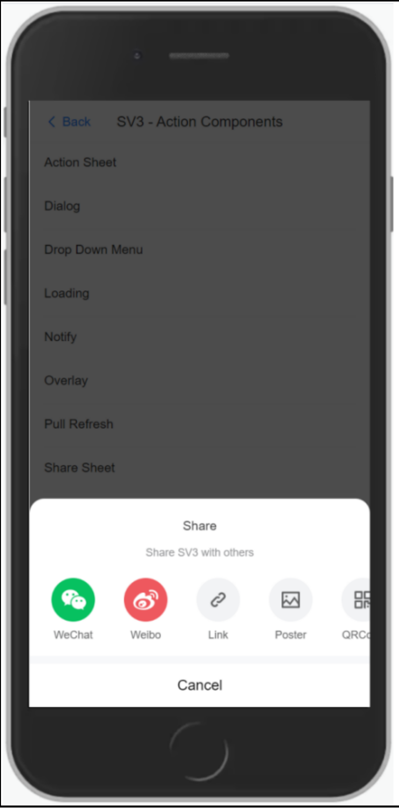
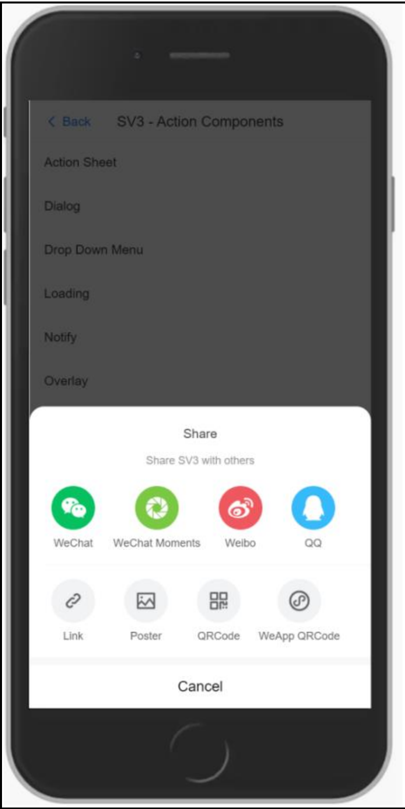
Used to provide interactive operations for pull-down refresh.

HTML Code	Output

```
<van-pull-refresh v-model="loading"
@refresh="onRefresh">
  <p>Refresh Count: {{ count }}</p>
</van-pull-refresh>
```
	 Back SV3 - PullRefresh
	Refresh Count: 1

3.3.8 ShareSheet

A pop-up sharing panel at the bottom for displaying the action buttons corresponding to each sharing channel, without specific sharing logic.

HTML Code	Output
`<van-cell title="Show ShareSheet" @click="showShare = true" />` `<van-share-sheet` `v-model:show="showShare"` `title="Share"` `:options="options"` `@select="onSelect"` `/>`	
`<van-share-sheet v-model:show="showShare" title="Share" :options="options" />`	

3.3.9 SwipeCell

Used for cell components that can slide left and right to display operation buttons.

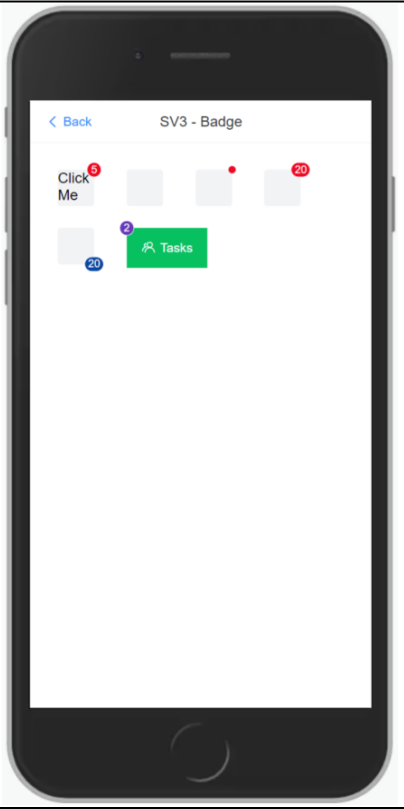
3.4 Display Components

3.4.1 Badge

Display a small badge or a red dot to the top-right of its child.

HTML Code	Output

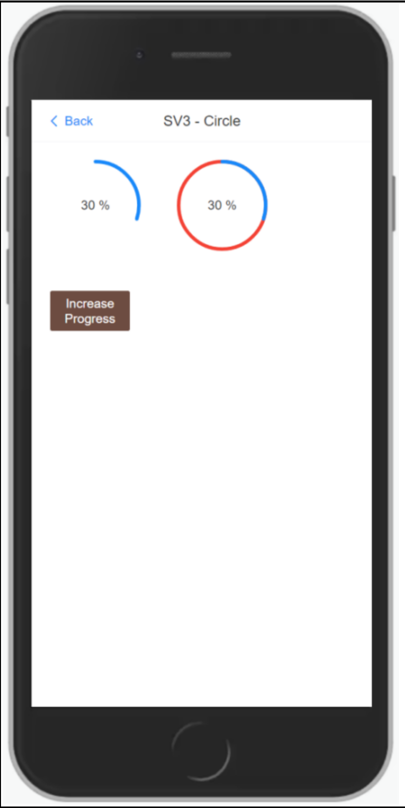
```html
<van-badge :content="5">
  <div class="child" />
</van-badge>
<van-badge :content="10">
  <div class="child" />
</van-badge>
<van-badge content="Hot">
  <div class="child" />
</van-badge>
<van-badge dot>
  <div class="child" />
</van-badge>

<style>
  .child {
    width: 40px;
    height: 40px;
    background: #f2f3f5;
    border-radius: 4px;
  }
</style>
```



3.4.2 Circle

Circular progress bar component, and supports gradient color animation.

HTML Code	Output

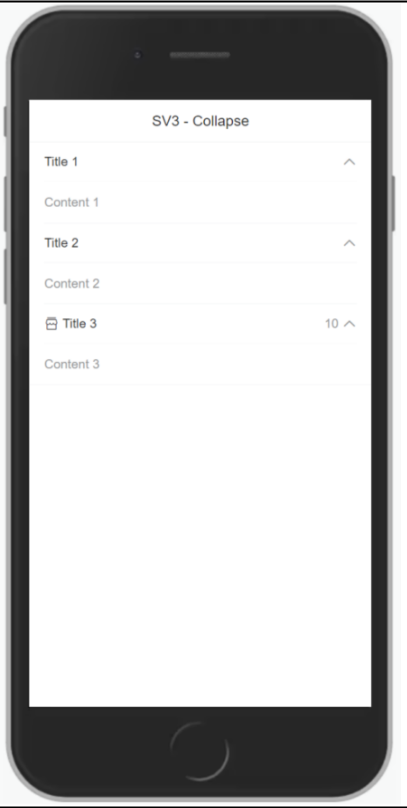
```
<van-circle
  v-model:current-rate="currentRate"
  :rate="30"
  :speed="100"
  :text="text"
/>
```
	

3.4.3 Collapse

Place a group of content in multiple collapsible panels, click the title of the panel to expand or contract its content.

HTML Code | Output

```
<van-collapse v-model="activeNames">
  <van-collapse-item title="Title1" name="1">Content
1</van-collapse-item>
  <van-collapse-item title="Title2" name="2">Content
2</van-collapse-item>
  <van-collapse-item title="Title3" name="3">Content
3</van-collapse-item>
</van-collapse>
```



SV3 - Collapse

Title 1

Content 1

Title 2

Content 2

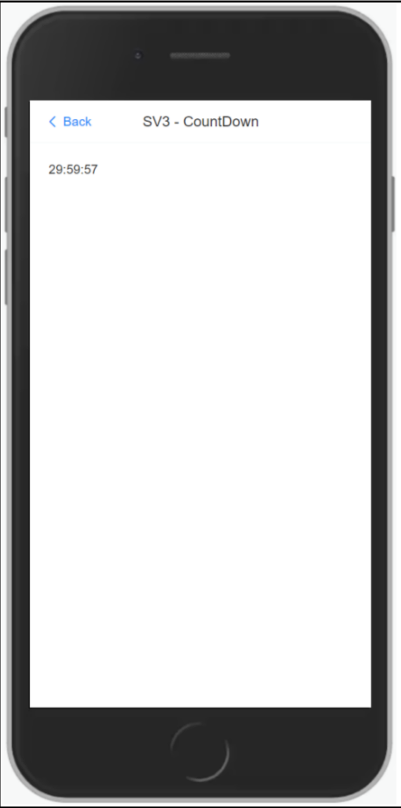
Title 3 10

Content 3

3.4.4 CountDown

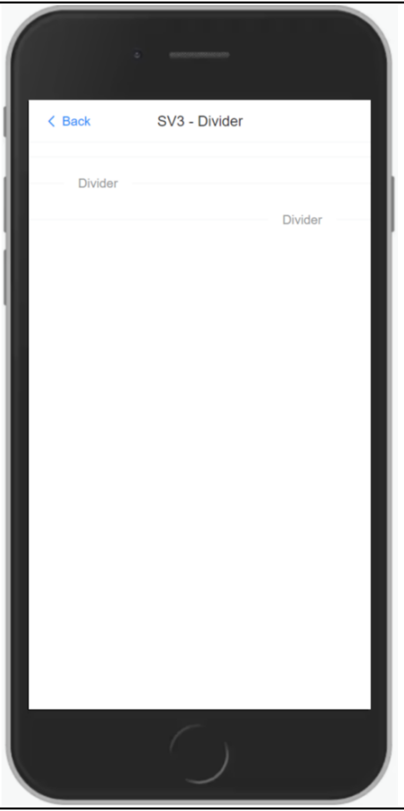
Used to display the countdown value in real time, and precision supports milliseconds.

HTML Code	Output

`<van-count-down :time="time" />`
`<van-count-down :time="time" format="DD Day, HH:mm:ss" />`
	 Back SV3 - CountDown 29:59:57

3.4.5 Divider

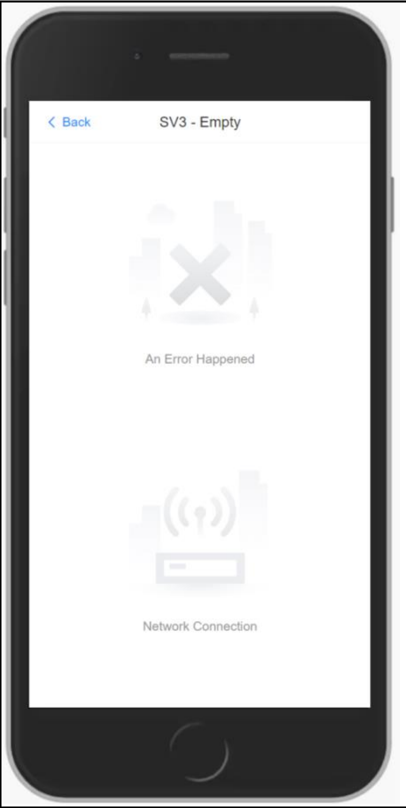
Separate content into multiple areas.

HTML Code	Output
`<van-divider>Text</van-divider>`	

3.4.6 Empty

Occupation reminder when empty.

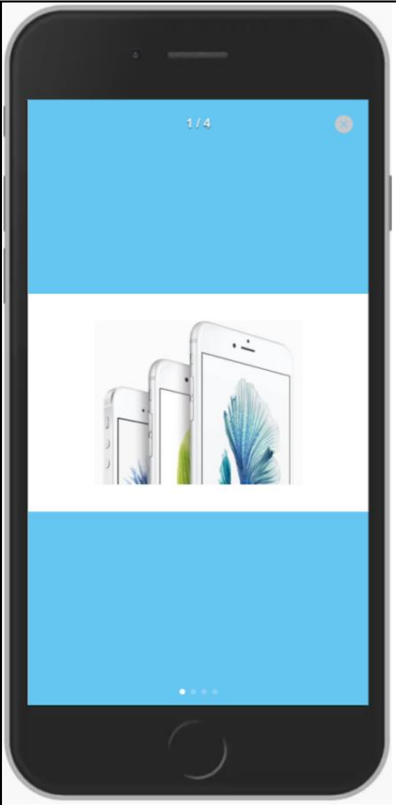
HTML Code	Output

```
<!-- Error -->
<van-empty image="error" description="Description" />
<!-- Network -->
<van-empty image="network" description="Description" />
<!-- Search -->
<van-empty image="search" description="Description" />
```
	 Back
	SV3 - Empty
	An Error Happened
	Network Connection

3.4.7 ImagePreview

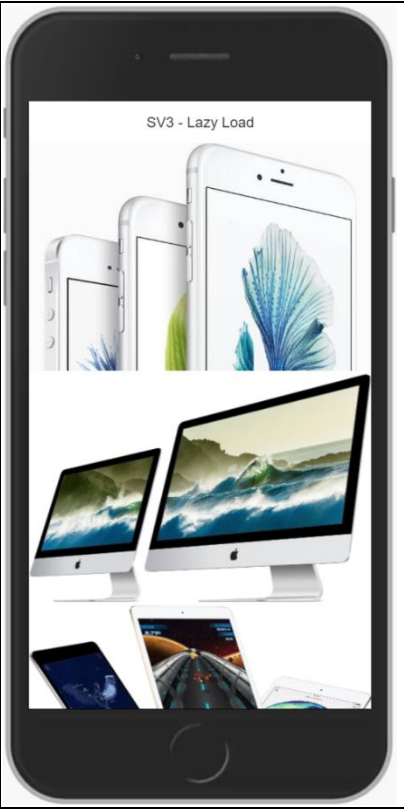
Used to zoom in and preview the picture, and it supports two methods: function call and component call.

HTML Code	Output

```
<van-image-preview v-model:show="show" :images="images"
@change="onChange">
  <template v-slot:index>Page: {{ index }}</template>
</van-image-preview>
```
	

3.4.8 Lazyload

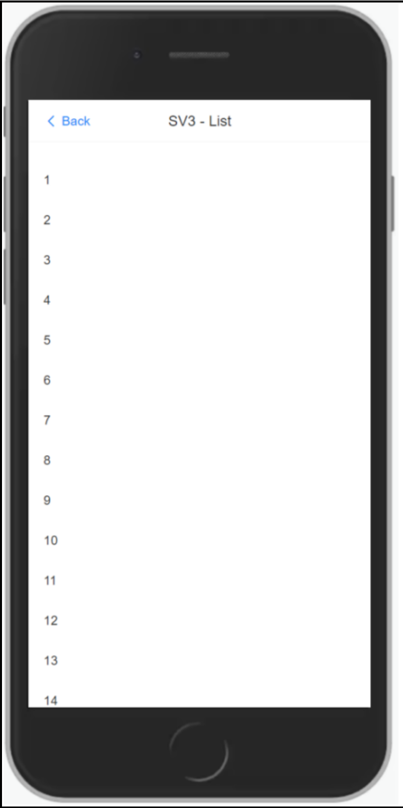
When the page needs to load a large amount of content, delay loading the content outside the visible area of the page to make the page load smoother.

HTML Code	Output
`<img v-for="img in imageList" v-lazy="img" />`	

3.4.9 List

A list component to show items and control loading status.

HTML Code	Output

```
<van-list
  v-model:loading="loading"
  :finished="finished"
  finished-text="Finished"
  @load="onLoad"
>
  <van-cell v-for="item in list" :key="item"
:title="item" />
</van-list>
```
	

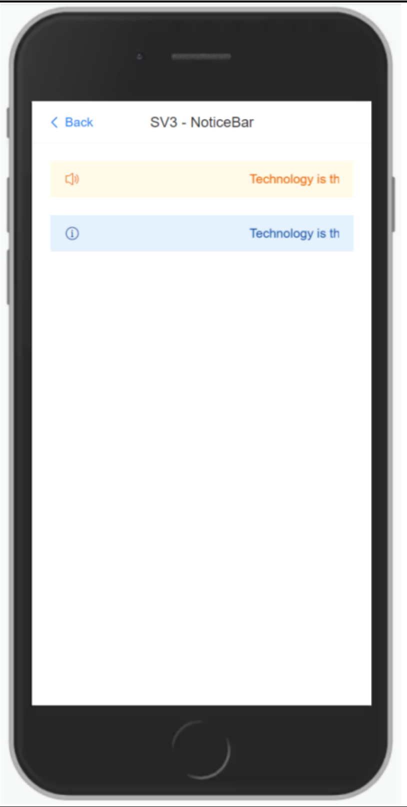
3.4.10 NoticeBar

Used to display a group of message notifications in a continuons loop.

HTML Code

```
<van-notice-bar
  text="Technology is the common soul of the people who 
developed it."
  left-icon="volume-o"
/>
```

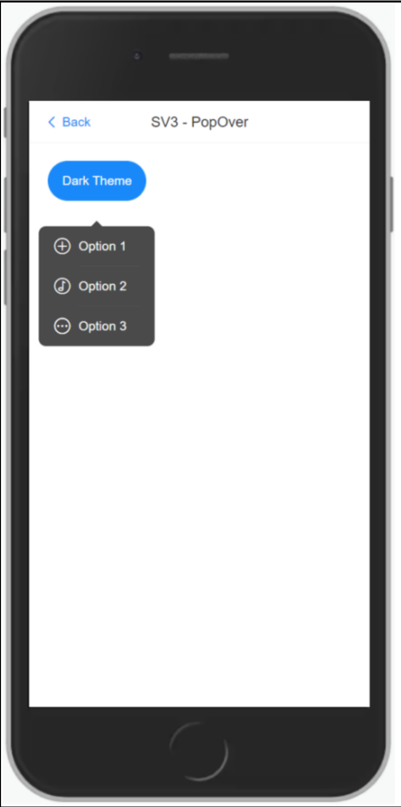
Output



3.4.11 Popover

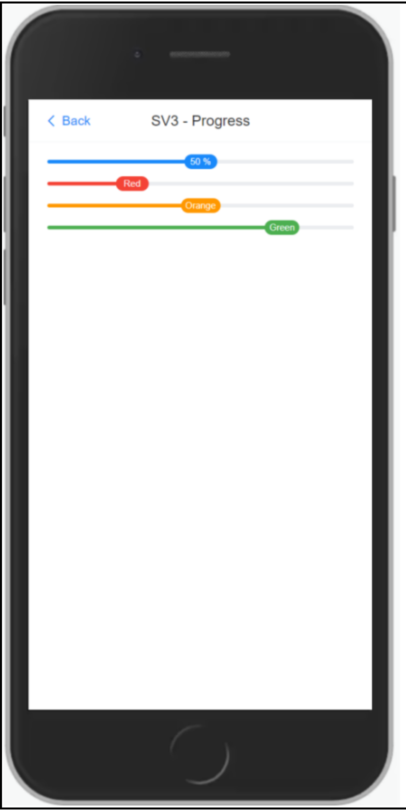
Used to display some content on top of another.

HTML Code	Output

`<van-popover v-model:show="showPopover"`
`:actions="actions" @select="onSelect">`
`  <template #reference>`
`    <van-button type="primary">Light Theme</van-button>`
`  </template>`
`</van-popover>`
	 Back
	SV3 - PopOver
	Dark Theme
	Option 1
	Option 2
	Option 3

3.4.12 Progress

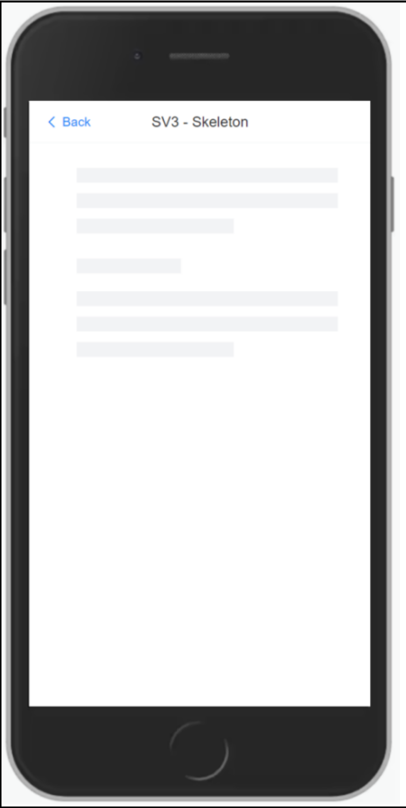
Used to show the current progress of the operation.

HTML Code	Output
`<van-progress :percentage="50" />`	

3.4.13 Skeleton

Used to display a set of placeholder graphics during the content loading process.

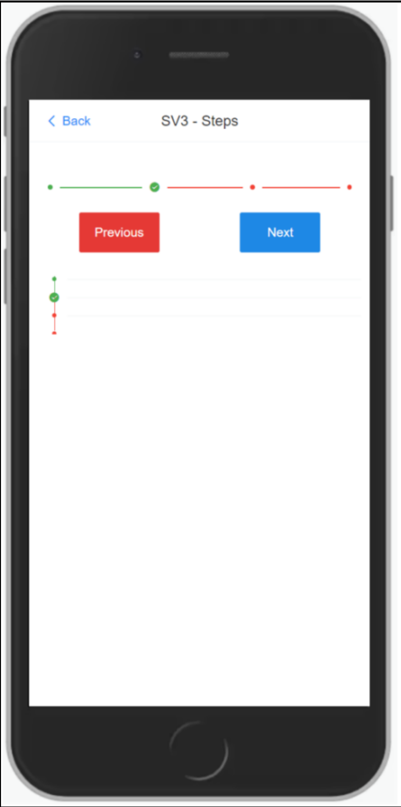
HTML Code	Output

`<van-skeleton title avatar :row="3" :loading="loading">`
`  <div>Content</div>`
`</van-skeleton>`
	

3.4.14 Steps

Used to show the various parts of the action flow and let the user know where the current action fits into the overall flow.

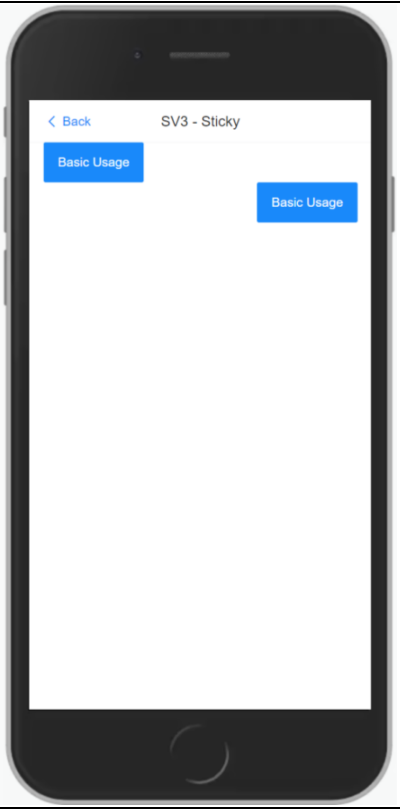
HTML Code	Output

```
<van-steps :active="active" active-icon="success"
active-color="#38f">
  <van-step>Step1</van-step>
  <van-step>Step2</van-step>
  <van-step>Step3</van-step>
  <van-step>Step4</van-step>
</van-steps>
```
	

3.4.15 Sticky

The sticky component is consistent with the effect achieved by the `position: sticky` property in CSS, in that when the component is within screen range, it will follow the normal layout arrangement, and when the component rolls out of screen range, it will always be fixed at the top of the screen.

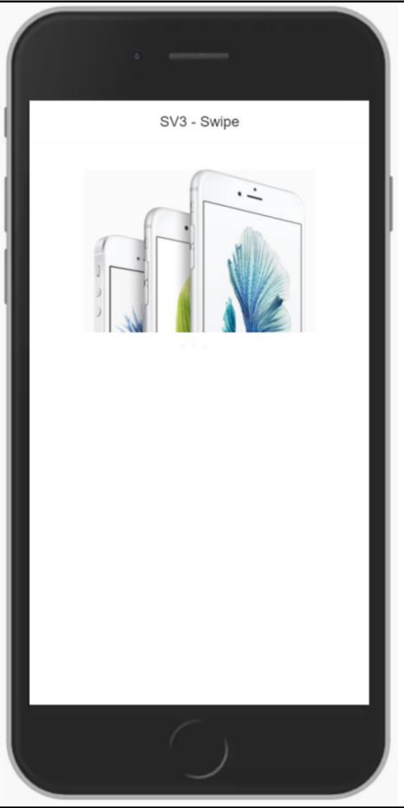
HTML Code	Output

`<van-sticky>`
`  <van-button type="primary">Basic Usage</van-button>`
`</van-sticky>`
	

3.4.16 Swipe

Used to loop a group of pictures or content.

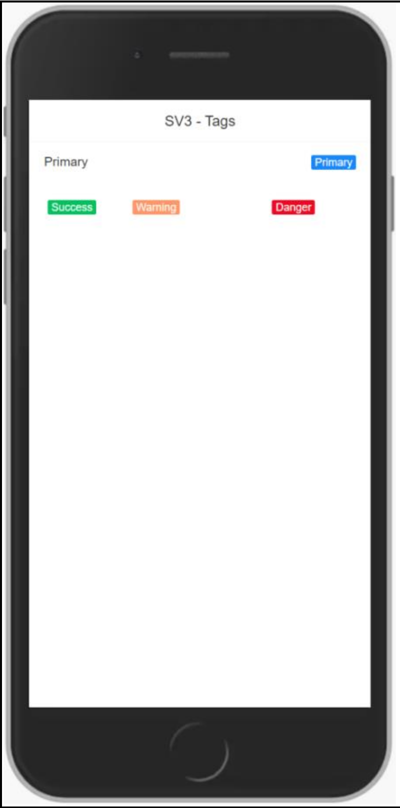
HTML Code	Output

```
<van-swipe :autoplay="3000" lazy-render>
  <van-swipe-item v-for="image in images" :key="image">
    <img :src="image" />
  </van-swipe-item>
</van-swipe>


<style>
  .my-swipe .van-swipe-item {
    color: #fff;
    font-size: 20px;
    line-height: 150px;
    text-align: center;
    background-color: #39a9ed;
  }
</style>
```
	

3.4.17 Tag

Used to mark keywords and summarize the main content.

HTML Code	Output

`<van-tag type="primary">Tag</van-tag>`
`<van-tag type="success">Tag</van-tag>`
`<van-tag type="danger">Tag</van-tag>`
`<van-tag type="warning">Tag</van-tag>`
	

3.5 Navigation Components

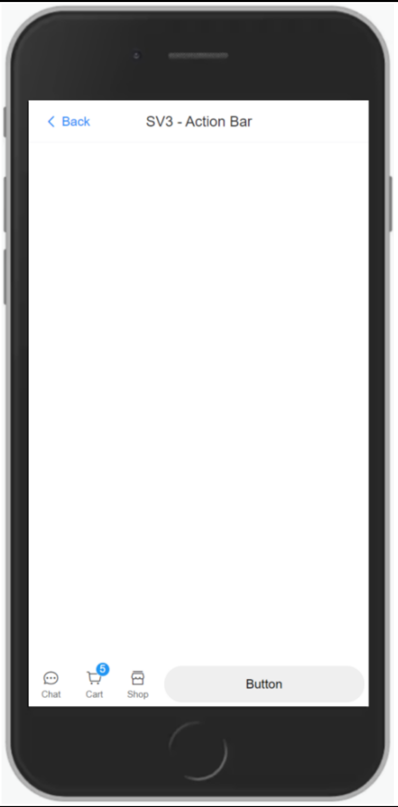
3.5.1 ActionBar

Used to provide convenient interaction for page-related operations.

HTML Code

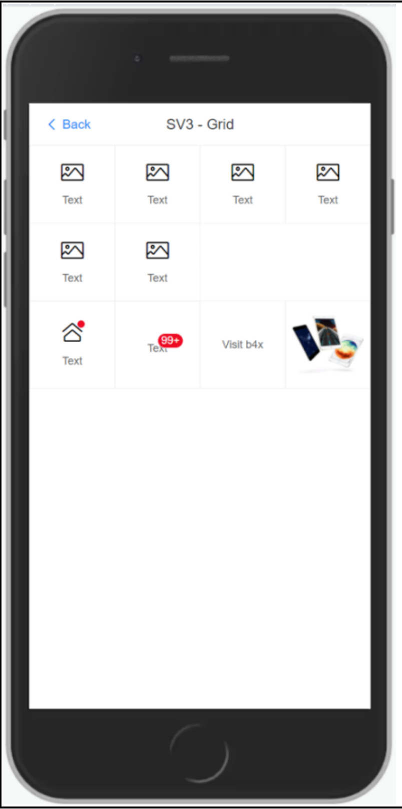
```
<van-action-bar>
  <van-action-bar-icon icon="chat-o" text="Icon1"
@click="onClickIcon" />
  <van-action-bar-icon icon="cart-o" text="Icon2"
@click="onClickIcon" />
  <van-action-bar-icon icon="shop-o" text="Icon3"
@click="onClickIcon" />
  <van-action-bar-button type="danger" text="Button"
@click="onClickButton" />
</van-action-bar>
```

Output



3.5.2 Grid

Used to divide the page into blocks of equal width in the horizontal direction for displaying content or page navigation.

HTML Code	Output
see code below	

```
<van-grid>
  <van-grid-item icon="photo-o" text="Text" />
  <van-grid-item icon="photo-o" text="Text" />
  <van-grid-item icon="photo-o" text="Text" />
  <van-grid-item icon="photo-o" text="Text" />
</van-grid>

<van-grid :border="false" :column-num="3">
  <van-grid-item>
    <van-image
src="https://cdn.jsdelivr.net/npm/@vant/assets/apple-
1.jpeg" />
  </van-grid-item>
  <van-grid-item>
    <van-image
src="https://cdn.jsdelivr.net/npm/@vant/assets/apple-
2.jpeg" />
  </van-grid-item>
  <van-grid-item>
    <van-image
src="https://cdn.jsdelivr.net/npm/@vant/assets/apple-
3.jpeg" />
  </van-grid-item>
</van-grid>
```

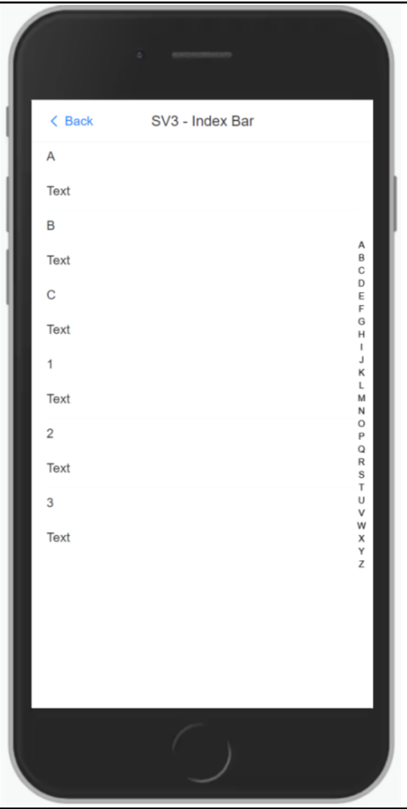
3.5.3 IndexBar

Used for indexed sorting display and quick positioning of lists.

HTML Code

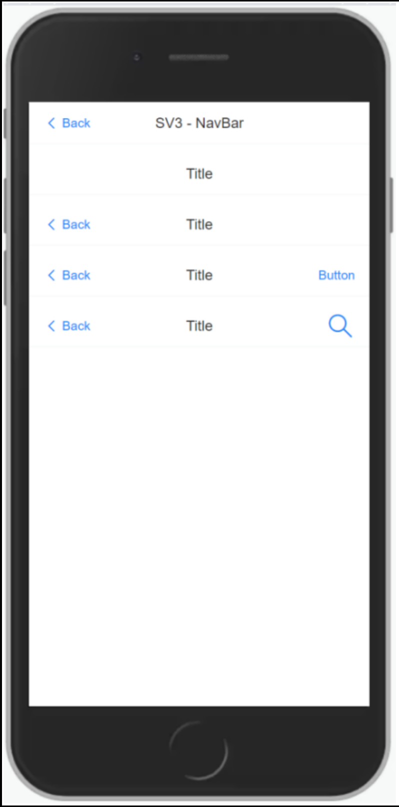
```html
<van-index-bar>
  <van-index-anchor index="A" />
  <van-cell title="Text" />
  <van-cell title="Text" />
  <van-cell title="Text" />

  <van-index-anchor index="B" />
  <van-cell title="Text" />
  <van-cell title="Text" />
  <van-cell title="Text" />

  ...
</van-index-bar>
```

Output



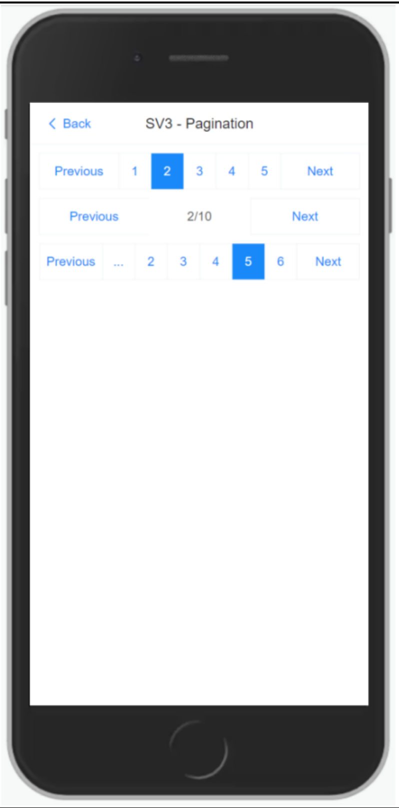
3.5.4 NavBar

Provide navigation function for the page, often used at the top of the page.

HTML Code	Output
`<van-nav-bar title="Title" left-text="Back" left-arrow @click-left="onClickLeft" />`	 Back SV3 - NavBar Title Back Title Back Title Button Back Title

3.5.5 Pagination

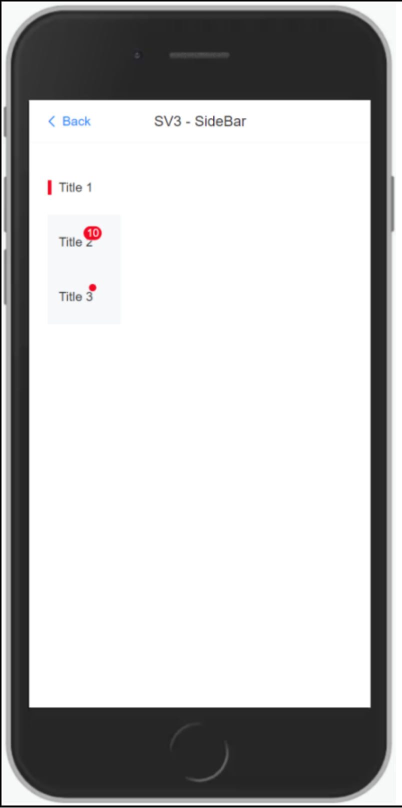
When the amount of data is too much, use pagination to separate the data, and load only one page at a time.

HTML Code	Output
`<van-pagination v-model="currentPage" :total-items="24" :items-per-page="5" />`	 Back SV3 - Pagination Previous 1 2 3 4 5 Next Previous 2/10 Next Previous ... 2 3 4 5 6 Next

3.5.6 Sidebar

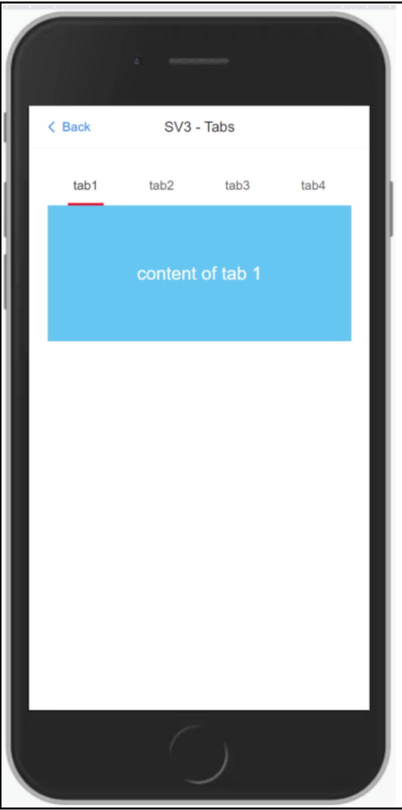
The vertically displayed navigation bar is used to switch between different content areas.

HTML Code	Output

```
<van-sidebar v-model="active">
  <van-sidebar-item title="Title" dot />
  <van-sidebar-item title="Title" badge="5" />
  <van-sidebar-item title="Title" badge="20" />
</van-sidebar>
```
	

3.5.7 Tabs

Used to switch between different content areas.

HTML Code	Output
`<van-tabs v-model:active="activeName">` `<van-tab title="tab 1" name="a">content of tab 1</van-tab>` `<van-tab title="tab 2" name="b">content of tab 2</van-tab>` `<van-tab title="tab 3" name="c">content of tab 3</van-tab>` `</van-tabs>`	

```
<van-tabs v-model:active="activeName">
  <van-tab title="tab 1" name="a">content of tab 1</van-tab>
  <van-tab title="tab 2" name="b">content of tab 2</van-tab>
  <van-tab title="tab 3" name="c">content of tab 3</van-tab>
</van-tabs>
```

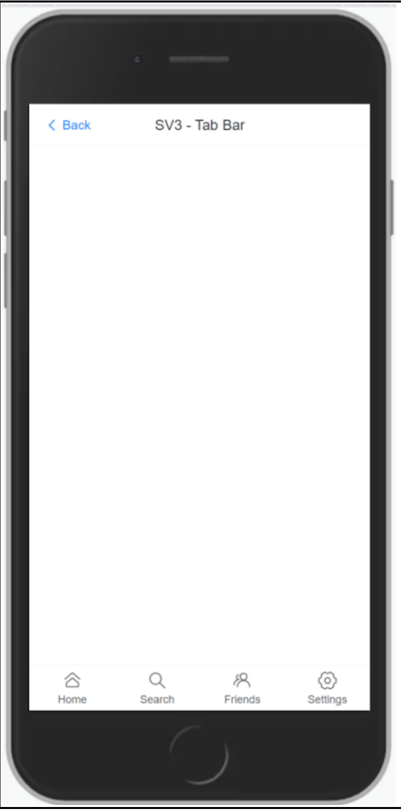
3.5.8 Tabbar

Used to switch between different pages.

HTML Code

```html
<van-tabbar v-model="active">
  <van-tabbar-item icon="home-o">Tab</van-tabbar-item>
  <van-tabbar-item icon="search">Tab</van-tabbar-item>
  <van-tabbar-item icon="friends-o">Tab</van-tabbar-
item>
  <van-tabbar-item icon="setting-o">Tab</van-tabbar-
item>
</van-tabbar>
```

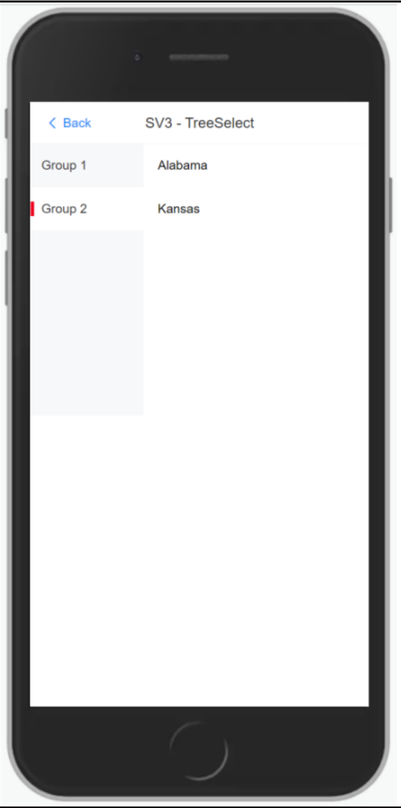
Output



3.5.9 TreeSelect

Used to select from a set of related data sets.

HTML Code	Output

```
<van-tree-select
  v-model:active-id="activeId"
  v-model:main-active-index="activeIndex"
  :items="items"
/>
```
	 Back
	SV3 - TreeSelect
	Group 1
	Group 2
	Alabama
	Kansas

3.6 Business Components

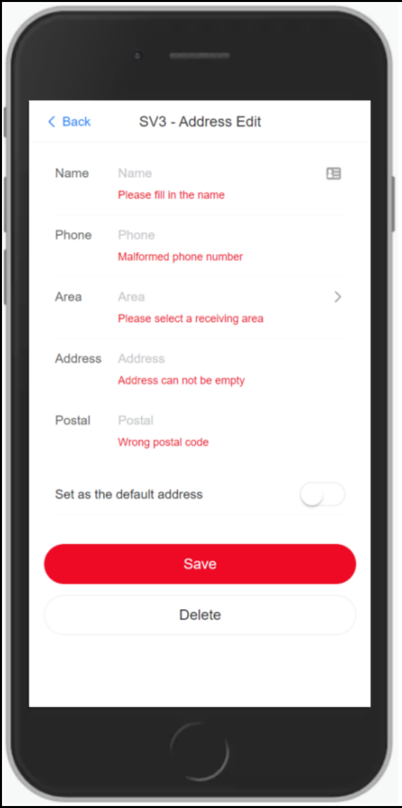
3.6.1 AddressEdit

Used to create, update, and delete receiving addresses.

HTML Code

```
<van-address-edit
  :area-list="areaList"
  show-postal
  show-delete
  show-set-default
  show-search-result
  :search-result="searchResult"
  :area-columns-placeholder="['Choose', 'Choose',
'Choose']"
  @save="onSave"
  @delete="onDelete"
  @change-detail="onChangeDetail"
/>
```

Output



< Back SV3 - Address Edit

Name: Name — Please fill in the name

Phone: Phone — Malformed phone number

Area: Area — Please select a receiving area

Address: Address — Address can not be empty

Postal: Postal — Wrong postal code

Set as the default address

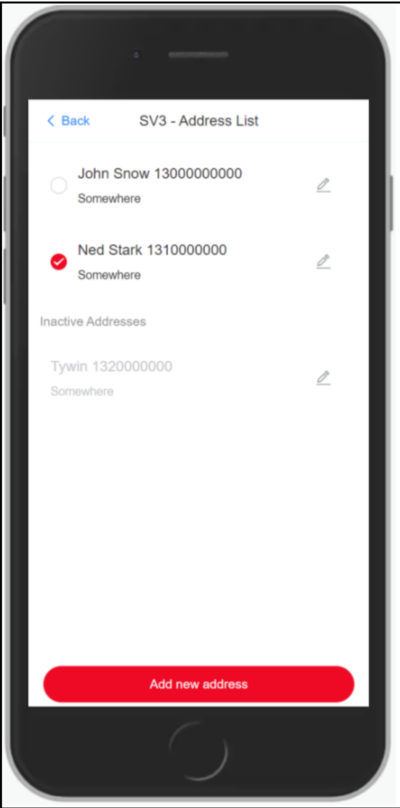
Save

Delete

3.6.2 AddressList

Display a list of receiving addresses.

HTML Code	Output

```
<van-address-list
  v-model="chosenAddressId"
  :list="list"
  :disabled-list="disabledList"
  disabled-text="The following address is out of range"
  default-tag-text="Default"
  @add="onAdd"
  @edit="onEdit"
/>
```
	

< Back SV3 - Address List

John Snow 13000000000
Somewhere

Ned Stark 1310000000
Somewhere

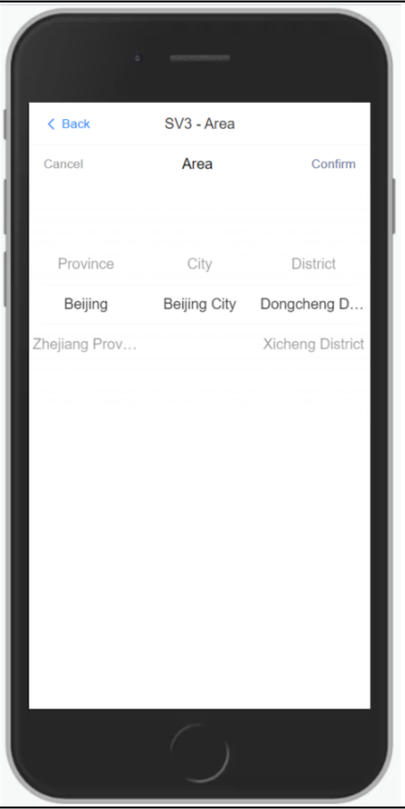
Inactive Addresses

Tywin 1320000000
Somewhere

Add new address

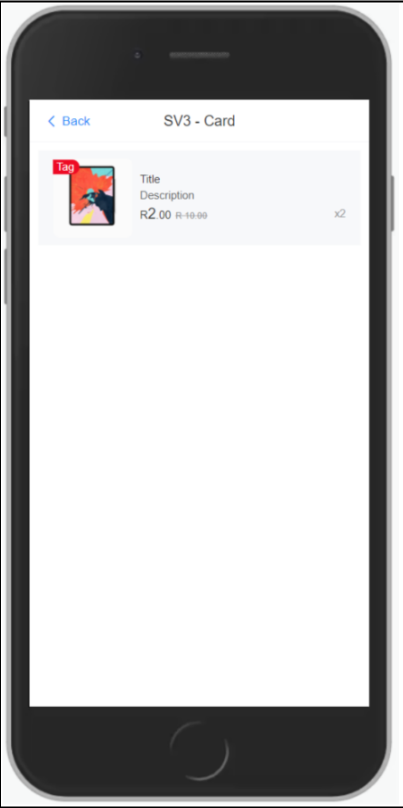
3.6.3 Area

A three-level linkage selection of provinces and cities, usually used in conjunction with Popup component.

HTML Code	Output
`<van-area title="Title" :area-list="areaList" />`	 Back SV3 - Area / Cancel Area Confirm / Province City District / Beijing Beijing City Dongcheng D… / Zhejiang Prov… Xicheng District

3.6.4 Card

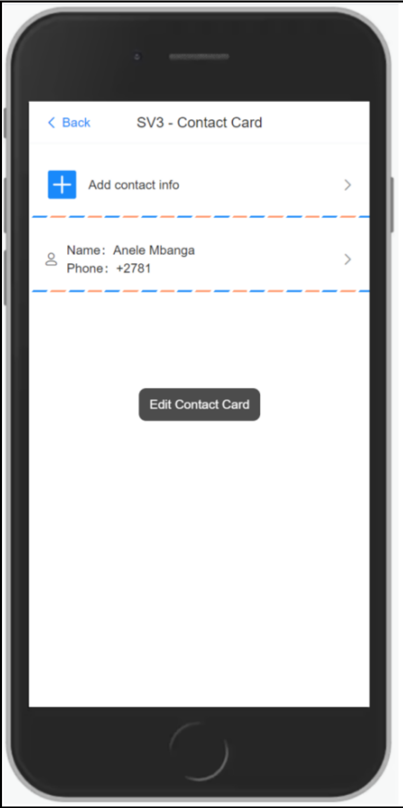
Used to display product pictures, prices and other information.

HTML Code	Output
(code below)	

```html
<van-card
  num="2"
  price="2.00"
  title="Title"
  desc="Description"

thumb="https://cdn.jsdelivr.net/npm/@vant/assets/ipad.jpeg"
/>
```

3.6.5 ContactCard

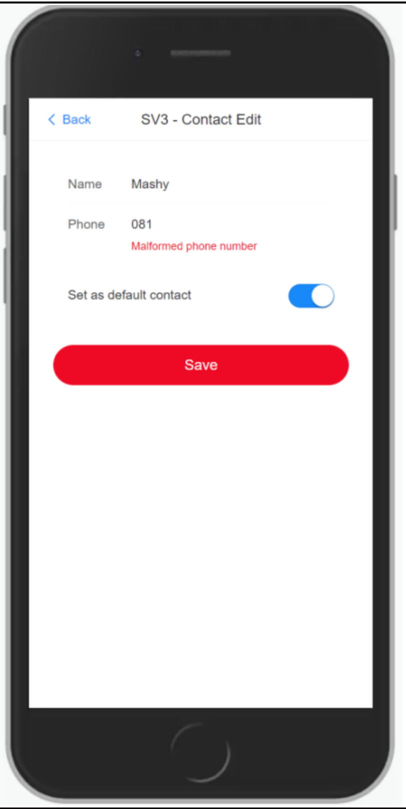
Display contact information in the form of cards.

HTML Code	Output
`<van-contact-card type="add" @click="onAdd" />`	 SV3 - Contact Card Add contact info Name： Anele Mbanga Phone： +2781

3.6.6 ContactEdit

Edit and save the contact information.

HTML Code	Output

```
<van-contact-edit
  is-edit
  show-set-default
  :contact-info="editingContact"
  set-default-label="Set as the default contact"
  @save="onSave"
  @delete="onDelete"
/>
```
	 Back SV3 - Contact Edit
	Name Mashy
	Phone 081
	Malformed phone number
	Set as default contact
	Save

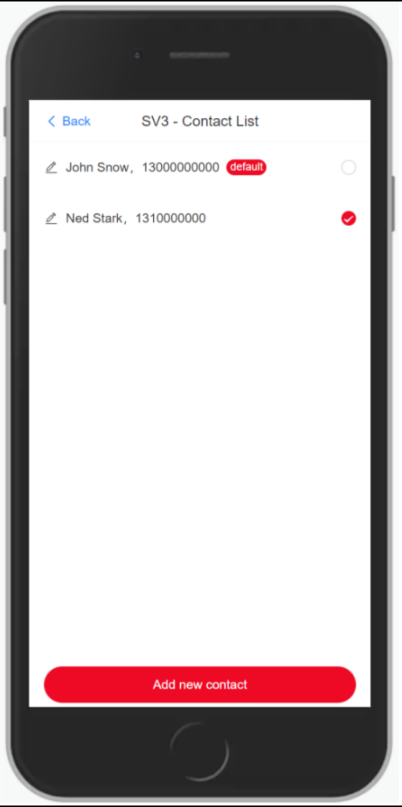
3.6.7 ContactList

Used to display the contact list.

HTML Code

```
<van-contact-list
  v-model="chosenContactId"
  :list="list"
  default-tag-text="default"
  @add="onAdd"
  @edit="onEdit"
  @select="onSelect"
/>
```

Output



< Back SV3 - Contact List

John Snow, 13000000000 default

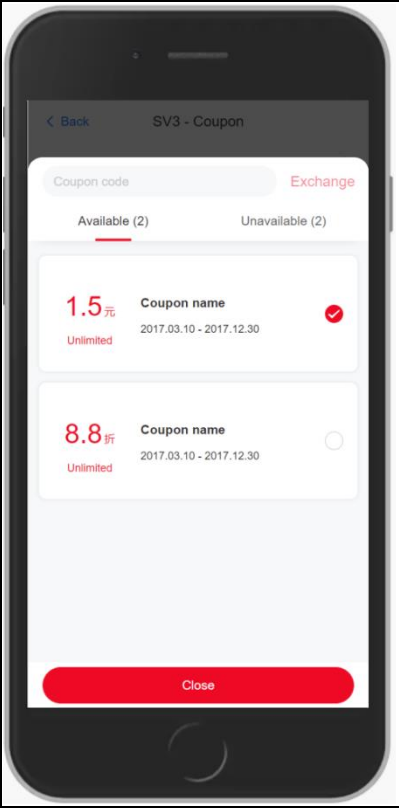
Ned Stark, 1310000000

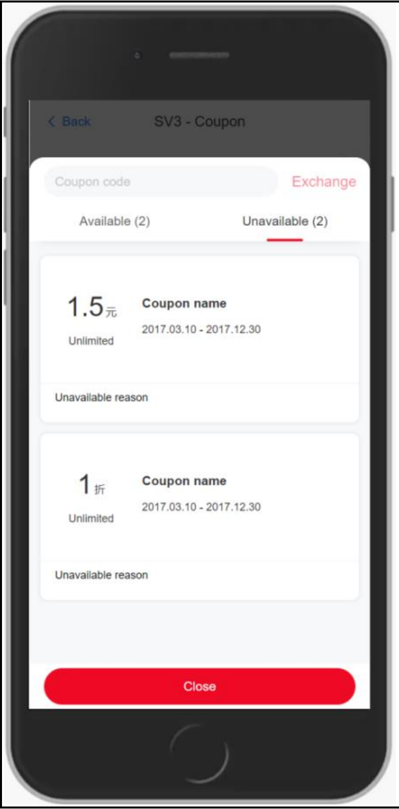
Add new contact

3.6.8 Coupon

Used for redemption and selection of coupons.

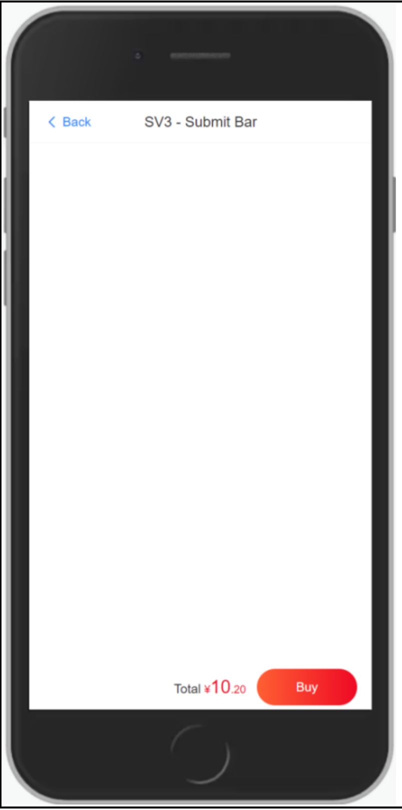
HTML Code	Output

`<!-- Coupon Cell -->`
`<van-coupon-cell`
`  :coupons="coupons"`
`  :chosen-coupon="chosenCoupon"`
`  @click="showList = true"`
`/>`
`<!-- Coupon List -->`
`<van-popup`
`  v-model:show="showList"`
`  round`
`  position="bottom"`
`  style="height: 90%; padding-top: 4px;"`
`>`
`  <van-coupon-list`
`    :coupons="coupons"`
`    :chosen-coupon="chosenCoupon"`
`    :disabled-coupons="disabledCoupons"`
`    @change="onChange"`
`    @exchange="onExchange"`
`  />`
`</van-popup>`
	 Back
	SV3 - Coupon
	Coupon code
	Exchange
	Available (2)
	Unavailable (2)
	1.5元
	Unlimited
	Coupon name
	2017.03.10 - 2017.12.30
	8.8折
	Unlimited
	Coupon name
	2017.03.10 - 2017.12.30
	Close

	 Back SV3 - Coupon Coupon code Exchange Available (2) Unavailable (2) 1.5元 Unlimited Coupon name 2017.03.10 - 2017.12.30 Unavailable reason 1折 Unlimited Coupon name 2017.03.10 - 2017.12.30 Unavailable reason Close

3.6.9 SubmitBar

Used to display the order amount and submit the order.

HTML Code	Output
`<van-submit-bar :price="3050" button-text="Submit" @submit="onSubmit" />`	 Back SV3 - Submit Bar Total ¥10.20 Buy