Development guide for Widgets in Skolon

What are widgets in Skolon?

Widgets in Skolon are small and simple apps that users can quickly view and interact with. The widgets are loaded in a sandboxed iframe, which results in some restrictions on what can be done. The only permissions given are allow-scripts and allow-same-origin. Some of the things blocked by the sandbox can however be done using Skolons SDK.
In this guide, you will learn more about how to get started with your widget and how to use Skolons SDK.

How to build a widget
- Import the SDK and get started with your widget
  - Import the SDK
  - Hello world in all sizes of the widget
- How to use the SDK
  - The openTab function
  - The init function
  - The openModal function
  - Get value from modal when closing it

How to build a widget

Choose a framework of your choice. At Skolon we use the Vue framework to build widgets.
In order for the widget to be able to interact with Skolon, we have developed a Software Development Kit (SDK). The SDK provides an API to interact with Skolon through a widget.

Import the SDK and get started with your widget

Once you have created your project, it's time to import the SDK and start developing your widget.

Import the SDK

1. To import the SDK, run the command **yarn add @skolon/widget-sdk** or **npm install @skolon/widget-sdk** in your terminal.
2. Import the SDK into your code

```
1  <script setup lang="ts">
2  import { init } from "@skolon/widget-sdk";
3
4  </script>
5
6  <template>
7    <main></main>
8  </template>
9
```

Hello world in all sizes of the widget

After the SDK is imported, we add Hello world to our template and start our local development server.

```
<script setup lang="ts">
import { init } from "@skolon/widget-sdk";

</script>

<template>
  <p>Hello World</p>
</template>

```

At the local address that Vue set up for us we can now see that Hello world is printed. It's this page we want to display in our widget.

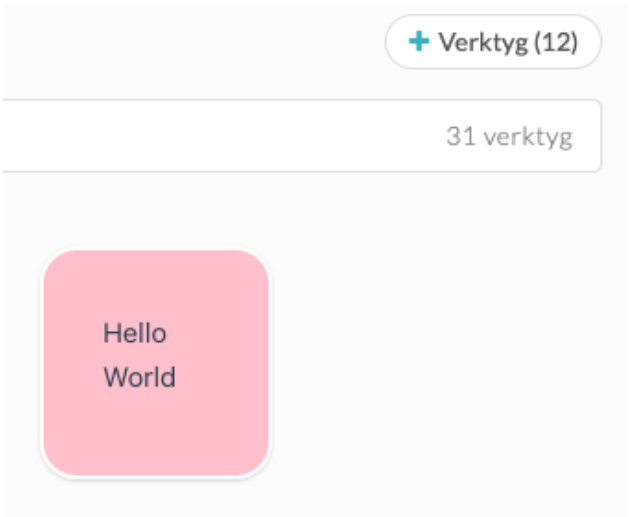
In Skolon, it's possible to display the widget in a sandbox environment. We start the sandbox environment and load our page (the one above) into a widget. Before we see the result, let's have a look at the measurements of the widget's different sizes.

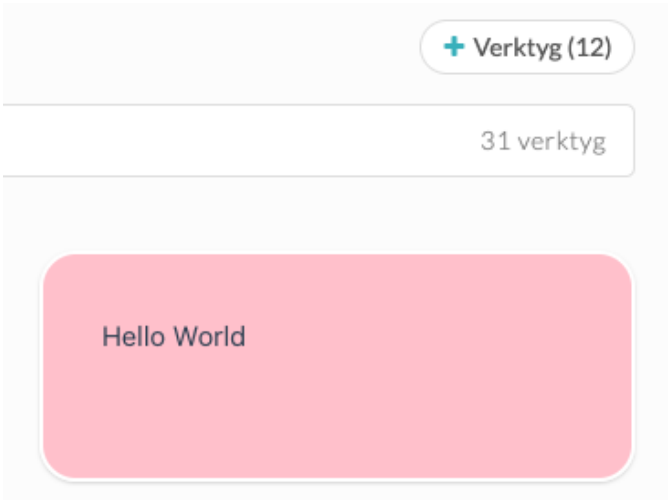
The widget comes in three different sizes, small, medium and large.

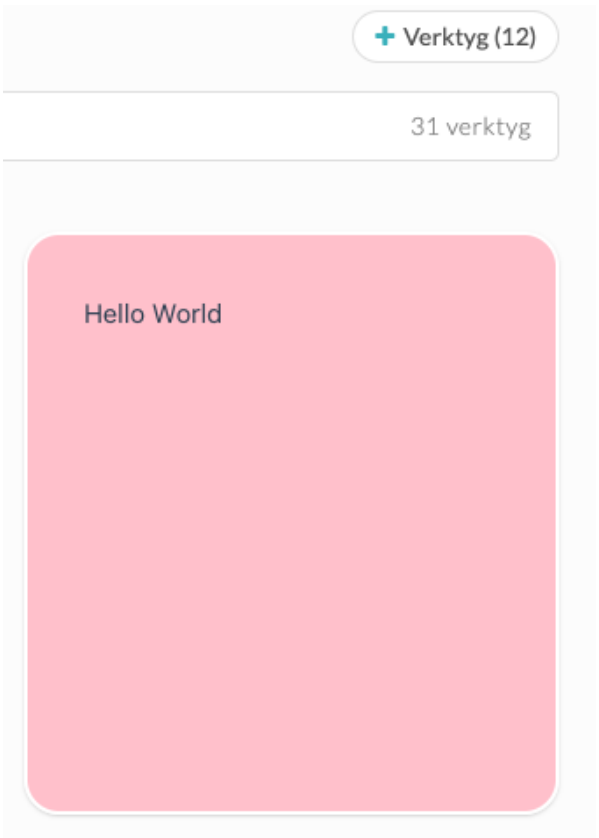
Widget	**Width (in pixels)**	**Height (in pixels)**
Small	122	122
Medium	314	122
Large	314	314

The width may vary depending on the screen size, but what is listed above is the minimum width.

Now, let's take a look at how the widget, in each size, displays in the sandbox environment.







How to use the SDK

With the help of the SDK, we can e.g. open a modal, open a link in a new tab or get information about the size of the widget. You can find more information on how the SDK works and what features it provides in this link: https://www.npmjs.com/package/@skolon/widget-sdk
In this section, you will find a number of examples of how to use the SDK when developing a widget.

The openTab function

The openTab function makes it possible to add a link to another page in the widget which, when clicked, opens in a new tab in the user's browser. See code example below.

```
<script setup lang="ts">
import { openTab } from "@skolon/widget-sdk";

const url = "https://skolon.com";

function onClick() {
  openTab(url);
}
</script>

<template>
  <button @click="onClick()">Click here to open link</button>
</template>
```

The init function

The init function can for example provide information about the size of the widget. widgetSize.width specifies how many columns the widget takes up and widgetSize.height specifies how many rows the widget spans.
In the table below we can see how many columns and rows each widget size spans over. This is the same value that widgetSize.width and widgetSize.height in the init function gives us.

Widget	Columns (widgetWidth)	Rows (widgetHeight)
Small	1	1
Medium	2	1
Large	2	2

The information about the widget's size can be useful if, for example, you want to adapt the design in your code according to the size of the widget.
See code example on how the init function is called and how it can be used.

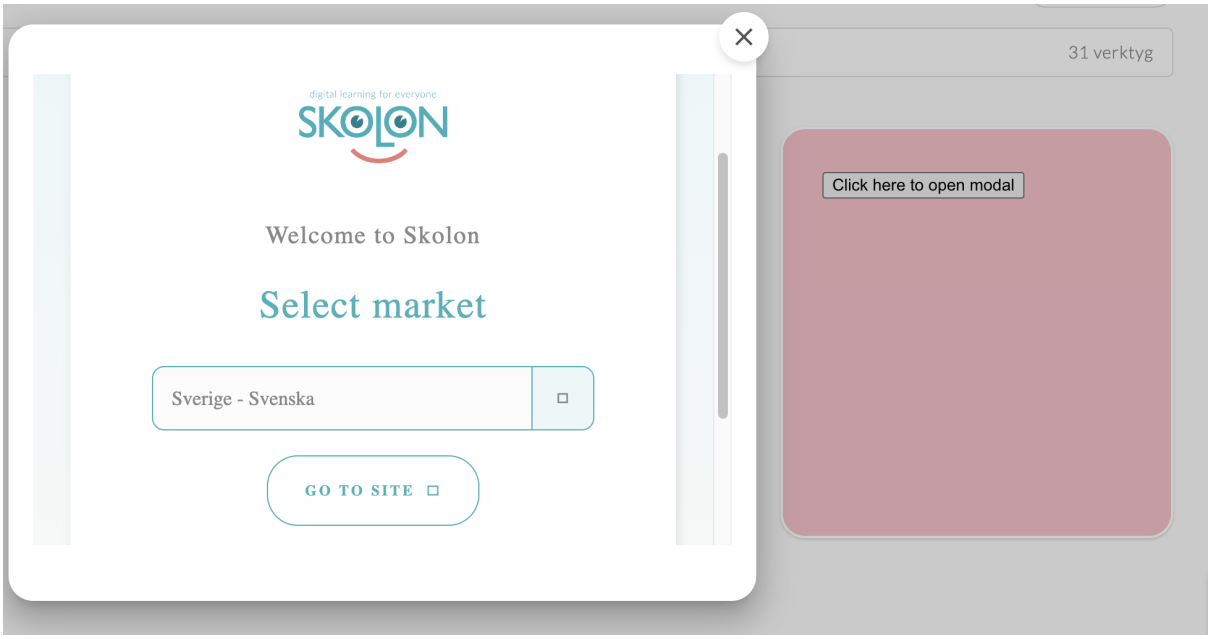
```
<script setup lang="ts">
import { init } from "@skolon/widget-sdk";
import { computed, reactive } from "vue";

interface State {
  widgetWidth: number | null;
  widgetHeight: number | null;
}

const state: State = reactive({
  widgetWidth: null,
  widgetHeight: null,
});

init().then((response) => {
  state.widgetWidth = response.widgetSize.width;
  state.widgetHeight = response.widgetSize.height;
});

const widgetSize = computed(() => {
  let size = "";

  if (state.widgetWidth == 1 && state.widgetHeight == 1) {
    size = "smallWidget";
  }
  if (state.widgetWidth == 2 && state.widgetHeight == 1) {
    size = "mediumWidget";
  }
  if (state.widgetWidth == 2 && state.widgetHeight == 2) {
    size = "largeWidget";
  }

  return size;
});
</script>

<template>
  <p>{{ widgetSize }}</p>
</template>
```

The openModal function

The openModal function opens a modal that displays an iframe for the specified url.
See code example.

```
<script setup lang="ts">
import { openModal } from "@skolon/widget-sdk";

function onClick() {
  openModal("http://skolon.com");
}
</script>

<template>
  <button @click="onClick()">Click here to open modal</button>
</template>

```

After clicking the “Click here to open modal” button, the modal opens and displays the content from the url specified in the openModal function (in our example http://skolon.com)



Get value from modal when closing it

In the section above, we looked at the openModal function. In the SDK, there is also a closeModal function. The closeModal function makes it possible to send data between the modal to the widget. This function is useful if, for example, the user has to make a choice in the modal which affects what the widget should display.

In the code example below we have created a new Vue file called Modal.vue.
In this file we have implemented code that will be shown in the modal. We have a text that says "Hello from modal" and a button with the text "Close Modal". We have also implemented the closeModal function inside the onClick function. In the closeModal function we are passing the variable "sendToWidget" which contains the string "You closed the modal". This variable will later on be passed to the widget.

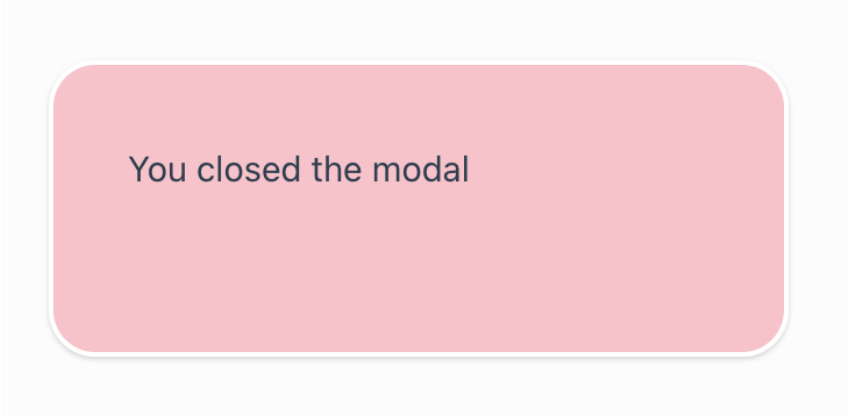
```vue
<script setup lang="ts">
import { closeModal } from "@skolon/widget-sdk";

var sendToWidget = "You closed the modal";

function onClick() {
  closeModal(sendToWidget);
}
</script>

<template>
  <p>Hello from Modal</p>
  <button @click="onClick()">Close modal</button>
</template>

```

When the user clicks on the "Close modal"-button the closeModal function is called.

The code example below shows the code of the widget. In this file we have extended the code for the openModal function. The URL in the openModal function now points to our Modal.vue file, which we looked at above. In addition to the changed URL, the code .then((response) => {}); is added. In this way we can receive the data that we sent with the closeModal function, which in our case is the string "You closed the modal".

```
<script setup lang="ts">
import { openModal } from "@skolon/widget-sdk";
import { reactive } from "vue";

const fromModal: String = reactive({
  responseString: null,
});

function onClick() {
  openModal("http://127.0.0.1:5173/Modal").then((response) => {
    fromModal.responseString = response.data;
  });
}
</script>

<template>
  <button v-if="fromModal.responseString == null" @click="onClick()">
    Click here to open modal
  </button>
  <span v-else>{{ fromModal.responseString }}</span>
</template>

```

This is how the widget will look after the user has clicked on the "Close modal"-button in the modal. The button with the text "Click here to open modal" is now gone. Instead we are showing the text that we got from the modal.



In the example above we are only sending a string from the modal to the widget. If more complex data is to be sent to the widget from the modal you may want to JSON encode the data. This can be done with JSON.stringify when sending data from modal, and JSON.parse when we receive the data.