miro | DE>ELOPERS

Miro Developer Platform

The Miro Developer Platform enables developers to create apps and integrations on top of Miro boards to add extra or custom functionality, and to sync Miro with third-party products.

Version 1.0.0 - Generated on July 9th, 2024

Table of Contents

Board Manipulation

Extension Points

Styling

Widgets Manipulation

Apps in Miro

Welcome to the Miro Developer Platform

> ### 📘 Public beta release of the Miro Developer Platform 2.0
>
> **Miro REST APIs and Web SDK 2.0 are available in public beta!**
> Explore a new dimension for Miro's infinite canvas.
>
> Note:
>
> - While Miro REST APIs and Web SDK 2.0 are in public beta, we recommend using them for development and testing.
> - For production and publication to the Marketplace, use Miro REST API and Web SDK 1.0.
>   Apps and integrations built with version 2.0 APIs and SDK are not compatible with version 1.0.

What you can do with Miro's API and SDKs

Build plugins into Miro interface

- Drag-and-drop plugins for visual asset management, like icons, images, etc
- 2-way integrations that would sync Miro items (like cards) with other tools, using custom fields, interfaces, etc
- Collaboration plugins for remote meetings, workshops, retrospectives, etc. These plugins could automate flow, hide widgets, create breakout rooms, etc

See web-plugins features →

Build 3rd party integrations with Miro data using REST API

- Read and update lists of users and their rights on specific boards
- Create and share boards
- Create, delete, move and update widgets
- Add custom data to widgets

See how to get access token and start using REST →

Embed Live Miro boards into other web apps using iFrame

- Provide an interface to choose a board a user has access to share via your app
- Embed a live collaborative board into a webpage

See how to embed boards →

To get started with any of the above you need to:

1. Create Dev team for Sandbox
2. Create an application and set it up (scopes, redirect_uri, web-plugin URL, etc)
3. Install the application to Dev Team to get **access token** or initialize **plugin**

Full guide here →

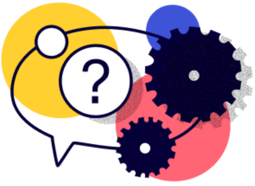
Learn more from our Developer Platform Team

- Ask questions in community
- See examples in our github repo

Getting started

Build your first Hello, World app

This is the start of your exciting journey with the Miro Developer Platform. Join us as we take you through the steps to build your first “Hello, World” Miro app. Once you’re acquainted with the steps to build an app, you’ll have the power to help teams collaborate smarter with custom apps and integrations.



Learn more about the Miro Developer Platform

Explore and learn more about the building blocks of the Miro Developer Platform. Get acquainted with fundamental concepts about Miro platform development.

Design a great app experience

When building apps, we all have a common goal—to make our users' lives more enjoyable and productive. We’ve compiled some design guidelines and best practices that dive into the details of building best-in-class Miro apps that help you provide a great user experience.

Build your first Hello, World app

This tutorial shows you how to get a simple app running in Miro. This app uses our Web SDK. If you’ve never built a Miro app before, you’re in the right place. Welcome, and let’s get started!
By the end of this guide, you will have an app that displays a sticky note, with the text **Hello, World**, on the board.

Prerequisites

Before you begin, ensure that you have the following prerequisites:

- You have a Miro account and you’ve signed in to Miro.

- You have Node 12.14 or higher installed.

Build your app

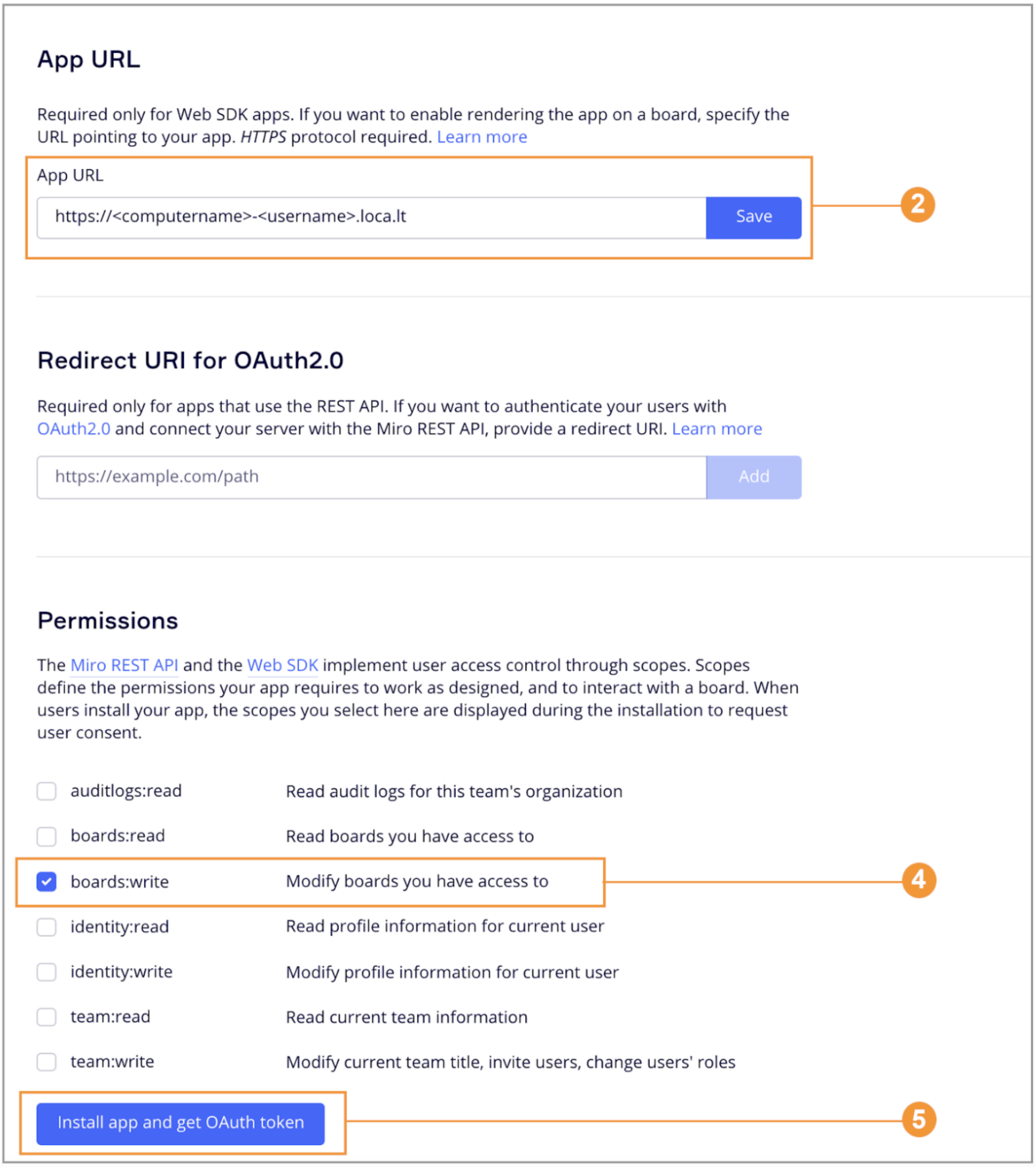
Here's a summary of the steps involved to build your first Hello, World app:

1. Bootstrap the Hello, World app.
2. Create a Developer team in Miro.
3. Create your app in Miro.
4. Configure and install your app in Miro.
5. Try out your app in Miro.

Configure and install your app in Miro

1. If you are on the **Hello world** app settings page, proceed to step 2.
   If you are not on the **Hello world** app settings page, go to your apps. On the **Created apps** section, click **Hello world**.
2. Scroll down to the **App URL** section.
3. In the **App URL** box, enter the URL you obtained in Step 3 of the Bootstrap the Hello, World app procedure, and then click **Save**.

> 📘 Note
>
> - The URL should look like this: **http://localhost:3000/**.
> - Ensure that you use an HTTPS URL.



App URL

Required only for Web SDK apps. If you want to enable rendering the app on a board, specify the URL pointing to your app. *HTTPS* protocol required. Learn more

App URL

https://<computername>-<username>.loca.lt

Redirect URI for OAuth2.0

Required only for apps that use the REST API. If you want to authenticate your users with OAuth2.0 and connect your server with the Miro REST API, provide a redirect URI. Learn more

https://example.com/path

Permissions

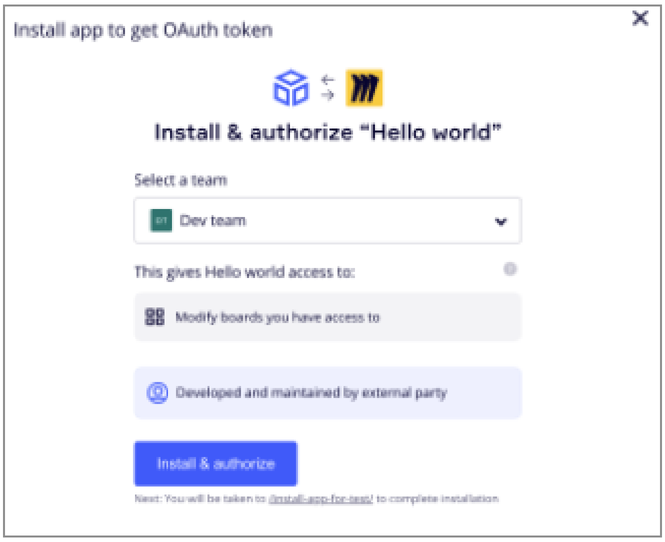
The Miro REST API and the Web SDK implement user access control through scopes. Scopes define the permissions your app requires to work as designed, and to interact with a board. When users install your app, the scopes you select here are displayed during the installation to request user consent.

□ auditlogs:read	Read audit logs for this team's organization
□ boards:read	Read boards you have access to
☑ boards:write	Modify boards you have access to
□ identity:read	Read profile information for current user
□ identity:write	Modify profile information for current user
□ team:read	Read current team information
□ team:write	Modify current team title, invite users, change users' roles

Figure 1: App settings

4. In the **Permissions** section, select the **boards:write** checkbox.
5. Click **Install app and get OAuth token**.
6. On the **Install app to get OAuth token** window:

a. In the **Select a team** list, click **Dev team**.



Install app to get OAuth token

Install & authorize "Hello world"

Select a team

Dev team

This gives Hello world access to:

Modify boards you have access to

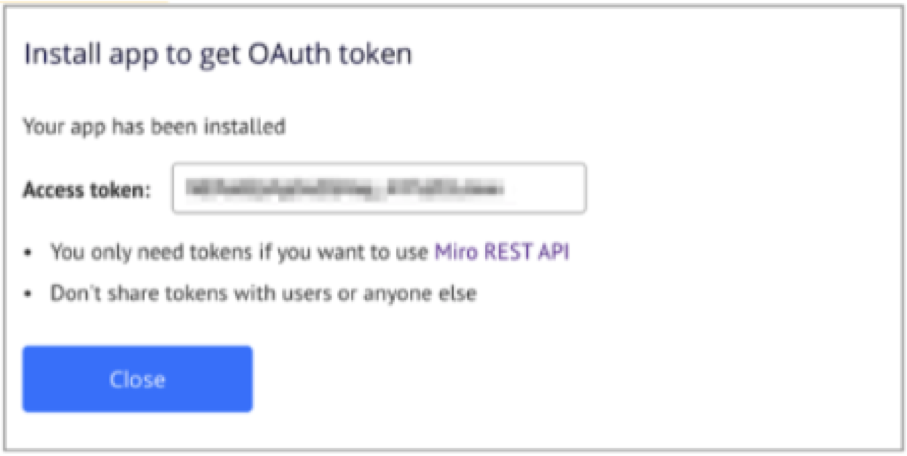
Developed and maintained by external party

Next: You will be taken to /install-app-for-test/ to complete installation.

Figure 2: Install & authorize app window

b. Click **Install & authorize**.

A message indicating that the app is successfully installed and your access token appears.



Install app to get OAuth token

Your app has been installed

Access token: [redacted]

- You only need tokens if you want to use Miro REST API
- Don't share tokens with users or anyone else

Figure 3: App installation successful and token window

c. Click **Close**.

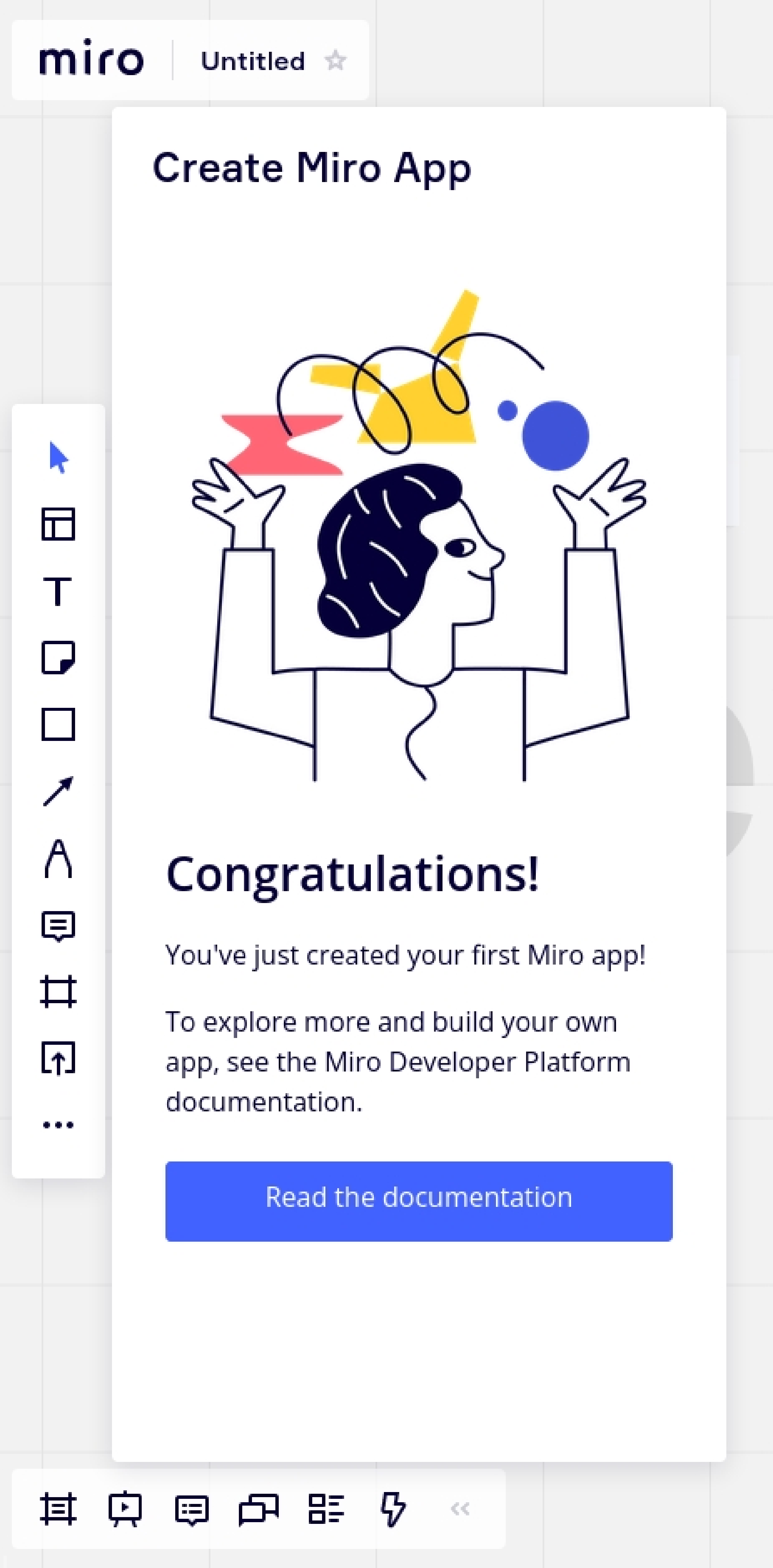
> **Note**
>
> You do not need to use the access token for the **Hello, World** app as it only uses the Miro Web SDK and it does not use the Miro REST API.

Try out your app in Miro

app' title screen shot.

Create Miro App

popup screen shot.



Create Miro App

Congratulations!

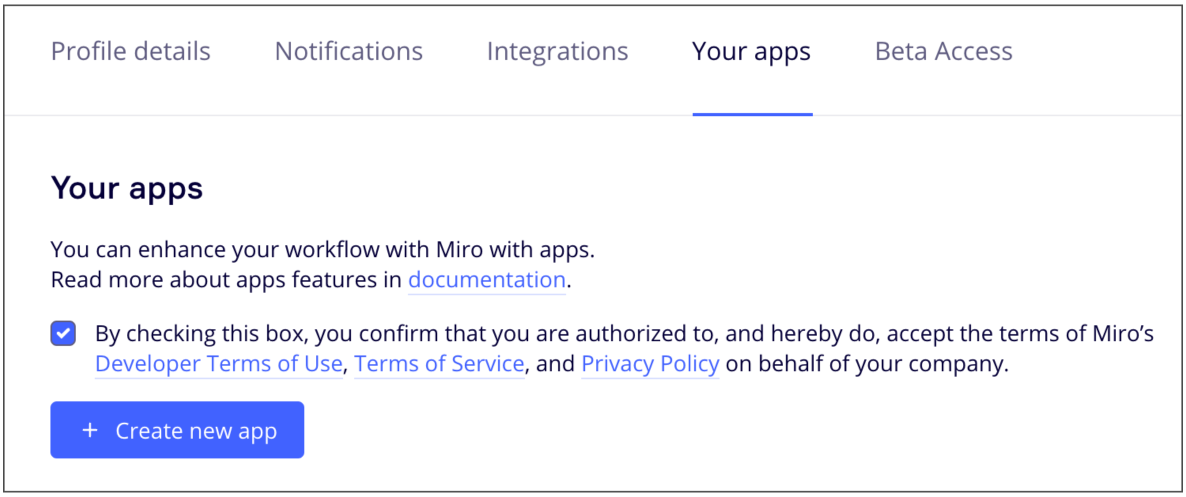
You've just created your first Miro app!

To explore more and build your own app, see the Miro Developer Platform documentation.

1. [Open your Miro dashboard](). Click the Dev team, and then click **New board**.
2. If the **Template Picker** appears, close the window.
3. Click >> or ... on the toolbar on the left side of the board, and then click **Create Miro App**.
   You should see a popup, with the title **Create Miro App** and text **Congratulations!** included, on the board.
   You did it! Congratulations on building your first Miro app. With the basics done, you can now start exploring more about the Miro Developer Platform.

Create your app in Miro

1. On the Your apps page, review the **Terms and Conditions**, select the **I agree to the Terms and Conditions** checkbox, and then click **Create new app**.



Profile details | Notifications | Integrations | Your apps | Beta Access

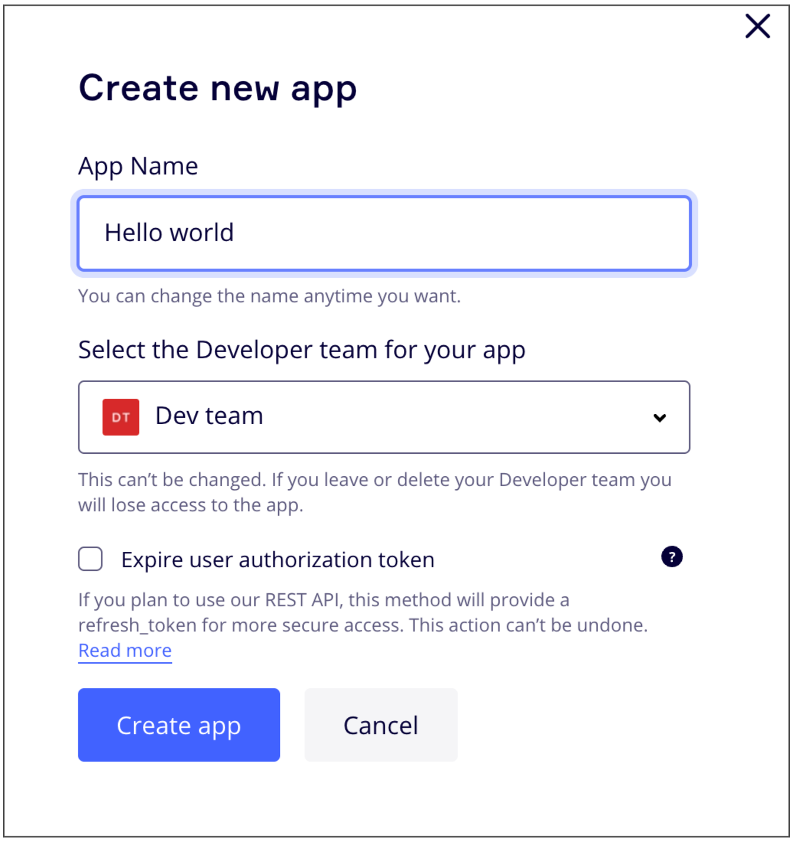
Your apps

You can enhance your workflow with Miro with apps.
Read more about apps features in documentation.

☑ By checking this box, you confirm that you are authorized to, and hereby do, accept the terms of Miro's Developer Terms of Use, Terms of Service, and Privacy Policy on behalf of your company.

Figure 1. Select the checkbox to accept the terms and conditions before proceeding to creating a new app.

2. On the **Create new app** window:
   a. In the **App name** box, enter **Hello world.**
   b. Click **Create app**.



Create new app

App Name: Hello world

You can change the name anytime you want.

Select the Developer team for your app: DT Dev team

This can't be changed. If you leave or delete your Developer team you will lose access to the app.

□ Expire user authorization token

If you plan to use our REST API, this method will provide a refresh_token for more secure access. This action can't be undone. Read more

Figure 2. On the **Create new app** modal, give the app a name and make it available for a team.

Bootstrap the Hello, World app

1. Open Terminal or the command prompt, and then run the following command:

```
$ npx create-miro-app@latest
```

For more information on the files, folder structure, and the app, click here.
2. Run your app locally.

```
$ cd my-miro-app
$ npm run start
```

3. Copy the app URL from the console output. The URL should look like this:

```
http://localhost:3000/
```

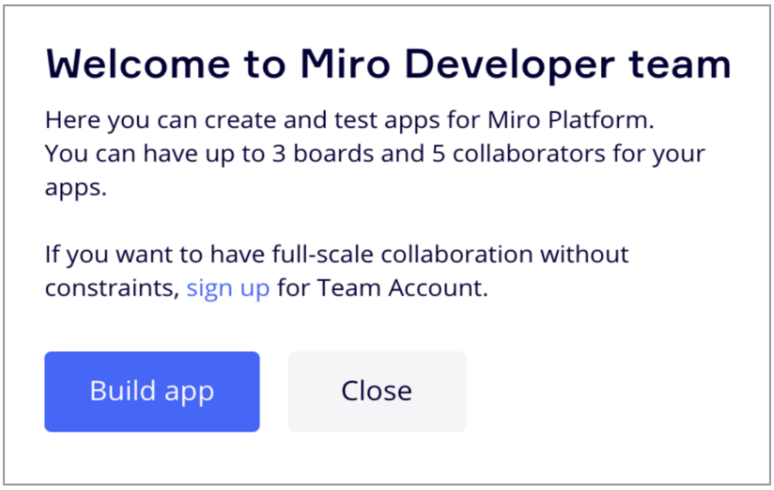
4. Open your browser, paste the URL you just copied in step 5, and press ENTER.
   A message indicating that your app is now running locally appears.

> 📘 Keep your app URL handy as you'll need this URL in a few minutes.

Create a Developer team in Miro

The Developer team is a special type of team that allows you to create and test apps without affecting your production boards, just like a sandbox with some limitations. Ensure that you use the Developer team only for development purposes, and not to complete any other type of work or remote collaboration.

1. Create a Developer team in Miro.
2. If you previously created the Developer team, the **Your apps** page appears instead of the **Welcome to Miro Developer team** box. Skip the steps in this procedure and proceed to Create your app in Miro.
   If you do not have the Developer team, clicking the link in step 1 creates a Developer team for you. Once the Developer team is successfully created, the **Welcome to Miro Developer team** box appears.



Welcome to Miro Developer team

Here you can create and test apps for Miro Platform. You can have up to 3 boards and 5 collaborators for your apps.

If you want to have full-scale collaboration without constraints, sign up for Team Account.

Figure 1: Welcome to Miro Developer team box

3. Click **Build app**, and then proceed to create your app in Miro.

📘 Note

A Developer team can have up to 3 boards and up to 5 collaborators. We recommend that you start testing your app on the Dev team, and if you need to test on more users, you can then proceed with installing your app on a team that supports your requirements. Consider a scenario where you want to create an app that eventually supports more than 50 users. As a start, you can create the app in the Dev team, test it with 3 to 5 users, and once all is well, install the app on a team that supports more than 50 users to see that everything is working as expected.

Differences between plans

Scopes available in all plans

`boards:read` — Read boards you have access to.

`boards:write` — Modify boards you have access to.

`team:read` — Read current team information.

`team:write` — Modify current team title, invite users.

`identity:read` — Read your profile information.

`identity:write` — Modify your profile information.

Scopes available only in Enterprise plan

`auditlogs:read` — Read audit logs for this team's organization

Examples:
When the app uses auditlogs:read scope, you can install the app only in the Enterprise plan team.
When the app uses boards:read scope, you can install the app for any team.

Developer team limitation

The developer team is a special team type for development purposes only. Do not use the Developer team to complete any other type of work or remote collaboration. The developer team limitation has the next limitations:

- Up to 5 users in a team
- Up to 3 boards in a team
- Board watermark
- No board export

👍 If you want to use Enterprise plan scopes but you are not Enterprise plan customer — contact us.

REST API

Quick Start

Create your app

Step-by-step of how to create a board and widget via REST API right from the documentation

1. Create Dev-team.
2. Create an app.
3. Configure scopes.
4. Get accessToken.
5. Create a board by clicking the `Try it` button in the documentation.
6. Create a card-widget on the board by clicking the `Try it` button.

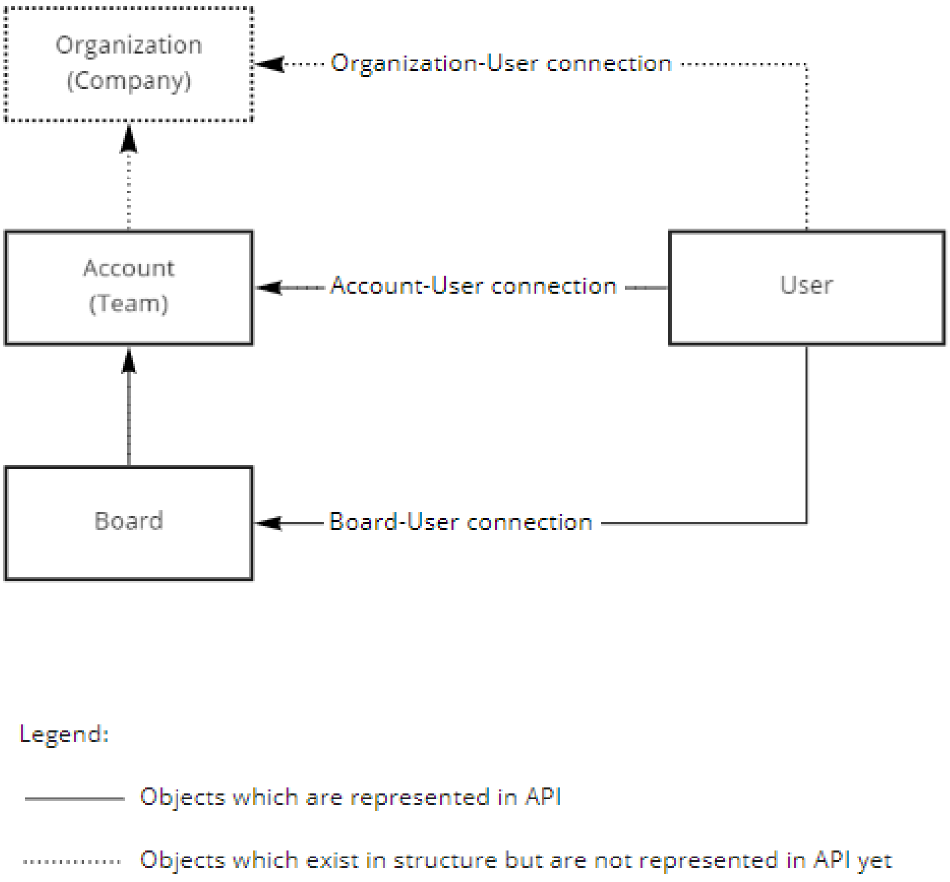
Check out our Demo App

We created NodeJS example to show how Miro Rest API works. This example includes:

- OAuth implementation
- REST API calls
- Serving static files for web-plugin

About API-related objects

- A user can participate in one or more teams, or even zero teams, if the user was deleted from all teams. For simplicity, you can assume that every user has at least one team.
- A team can belong to an organization (Company), and it may not. Organizations are not represented in the API yet.
- Every board belongs to exactly one team.
- Registered and invited users are connected with teams and boards via user-connection objects.


Organization
(Company)
Organization-User connection
Account
(Team)
Account-User connection
User
Board
Board-User connection
Legend:
Objects which are represented in API
Objects which exist in structure but are not represented in API yet

API Reference

Web-Plugins

Web-plugins overview

Web-plugins are a fast and easy way to create your own tools within Miro. With web-plugins you can add buttons to toolbars, render custom views in modals and sidebars, and modify objects on the board.

Web-plugins uses client-side technologies like HTML, JavaScript, and CSS. Server-side technologies like NodeJS or Java are not required for building web-plugins. You can use a tool or service of your choice to learn, build, and host apps.

Web-plugins are JS scripts that run on boards in isolated iframes. The JS code of a web-plugin communicates with the Miro application via PostMessage using the Miro SDK.

> 📘 SDK
>
> HTML
>
> ```html
> <script src="https://miro.com/app/static/sdk.1.1.js"></script>
> ```

Web-plugins SDK Reference

See the SDK Reference here.

Installation and authorization

Web-plugins can change the board content on behalf of the current user. A user must authorize a web-plugin via OAuth2.

For more details, see Web-plugin Authorization and Share your app.

Scopes

The permissions and capabilities of a web-plugin are defined by scopes, just as with the REST API. You must enable the required scopes in your application settings. When you enable a scope, each user has to grant your app the associated permissions.

boards:read — Read boards the user has access to

JavaScript

```javascript
miro.board.widgets.get()
miro.board.comments.get()
miro.board.groups.get()
miro.board.selection.get()
miro.board.selection.selectWidgets()
```

boards:write — Modify boards the user has access to

JavaScript

```javascript
miro.board.widgets.create()
miro.board.widgets.update()
miro.board.widgets.transformDelta()
miro.board.widgets.deleteById()
miro.board.widgets.bringForward()
miro.board.widgets.sendBackward()
```

Web-plugins features

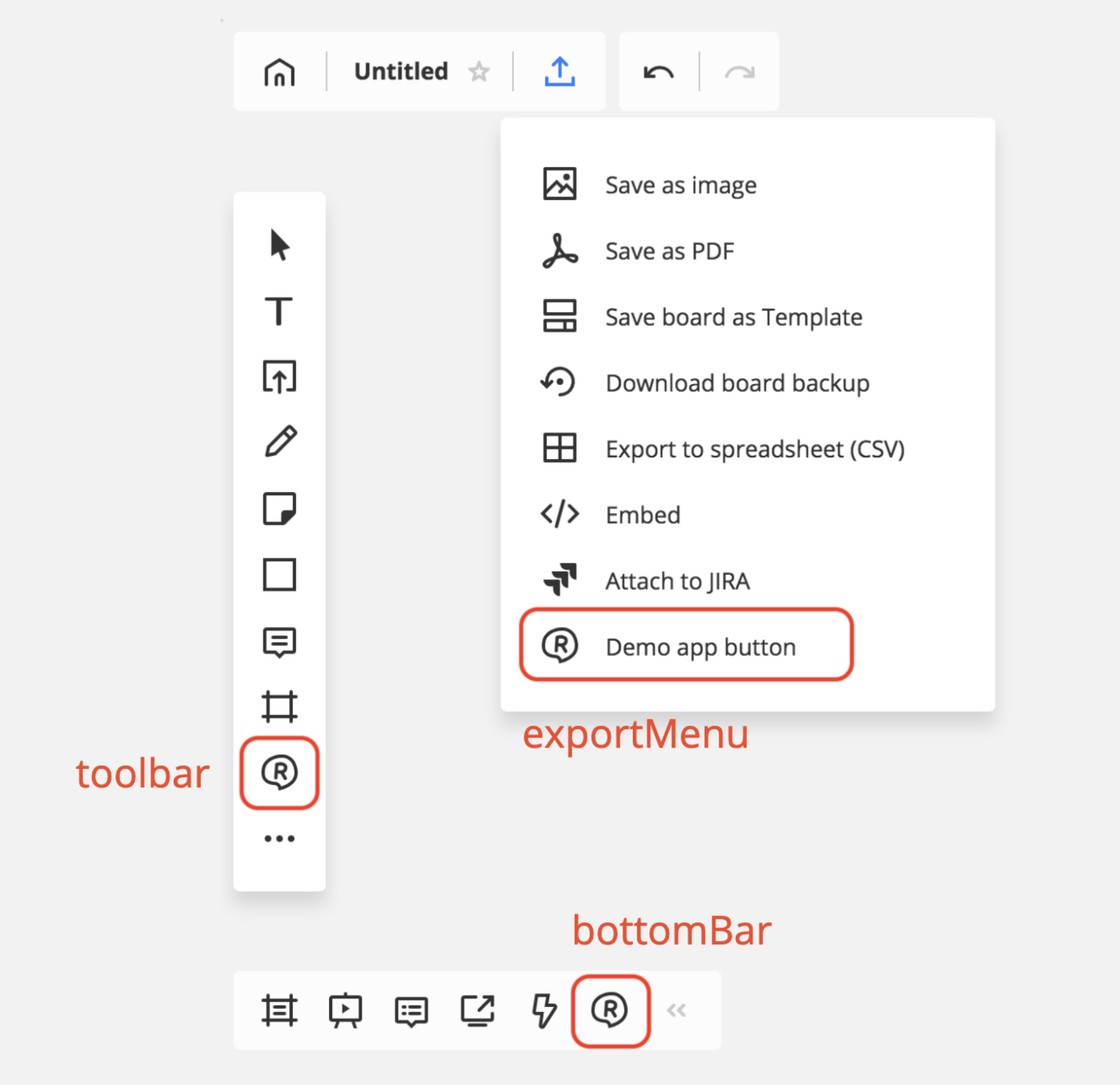
- Add buttons
- Render custom views
- Modify and read board content
- Modify and read tags on widgets
- Listen to events on the board
- Control the visibility of widgets for the current user
- Store metadata in a widget
- Disable editing on a widget
- Enable drag and drop of items from a custom view
- Control the visible area of the board
- Manage widget selection
- Display notifications
- Add options to Miro settings

Additional experimental methods are listed in the Web SDK reference. Names of experimental methods start with `__`. For example, `__getBoardUrlWithParams()`.

Add buttons

You can add buttons on different elements of the Miro interface. You can add buttons asynchronously after performing relevant checks. For example, you might choose to add an Edit button only for users who have the right to edit the board. This example illustrates how to add buttons asynchronously.

To apply a blue hover effect, use the SVG attribute `fill="currentColor".`



JavaScript

```
miro.onReady(() => {
  const icon24 = 'some svg here'

  miro.initialize({
    extensionPoints: {
      bottomBar: {
        title: 'Demo app button',
        svgIcon: icon24,
        onClick: () => {
          alert('Bottom bar item has been clicked')
        }
      },
    }
  })
})
```

Render custom views

You can open a *modal window*, *sidebar*, *bottom panel*, or *library* where you can render a custom view.

`miro.board.ui.openModal('custom-view.html')`
`miro.board.ui.openLeftSidebar('custom-view.html')`

All ui methods

```
miro.board.ui.openLeftSidebar(iframeURL: string, options?: {width?: number}): Promise<any>
miro.board.ui.openLibrary(iframeURL: string, options: {title: string}): Promise<any>
miro.board.ui.openModal(iframeURL: string, options?: {width?: number; height?: number}): Promise<any>
miro.board.ui.openBottomPanel(iframeURL: string, options?: {width?: number; height?: number}): Promise<any
miro.board.ui.closeLeftSidebar(data)
miro.board.ui.closeLibrary(data)
miro.board.ui.closeModal(data)
miro.board.ui.closeBottomPanel(data)
```

Modify and read board content

A web-plugin can read, create, and modify content on the board. An item of board content is contained in a "widget". The Miro SDK can perform operations on a set of supported widget types.

The web-plugin requires the `boards:read` scope to utilize methods that read board content, and the `boards:write` scope to modify board content.

Example

```
// Create new sticker
await miro.board.widgets.create({type: 'sticker', text: 'Hello'})

// Get all stickers on the board
let allStickers = await miro.board.widgets.get({type: 'sticker'})
console.log(allStickers)
```

All widgets methods

```
miro.board.widgets.create(widgets: {type: string; [index: string]: any}[]): Promise<IBaseWidget[]>
miro.board.widgets.get(filterBy?: Object): Promise<IBaseWidget[]>
miro.board.widgets.update(widgets: {id: string; [index: string]: any}[]): Promise<IBaseWidget[]>
miro.board.widgets.deleteById(widgetIds: InputWidgets): Promise<void>
miro.board.widgets.transformDelta(widgetIds: InputWidgets, deltaX: number | undefined, deltaY: number | un
miro.board.widgets.bringForward(widgetId: InputWidgets): Promise<void>
miro.board.widgets.sendBackward(widgetId: InputWidgets): Promise<void>
```

🚧 Rate limits

The Miro SDK enforces rate limits on widget operations. Each operation (create, update, or delete) on one widget costs one point. The current rate limit is 10,000 points per minute, summed over all widgets, and overall web-plugins, executing in a single client.

For example this operation costs 2 points:
miro.board.widgets.create([{type:'sticker', type:'sticker'}])

Modify and read tags on widgets

Stickers and card widgets can be tagged. These tags can be read and modified using the Miro SDK.

> ### 🚧 The API for tags is currently experimental
>
> Experimental APIs might change at any time.

API

```
interface ITag {
    id: string
    title: string
    color: string | number
    widgetIds: string[]
}

// Supported operations
miro.board.tags.create()
miro.board.tags.update()
miro.board.tags.delete()
miro.board.tags.get()
```

Example

```
// Create sticker and card with tag 'Red tag'
let widgets = await miro.board.widgets.create([
        {type: 'sticker', text: 'I am sticker'},
        {type: 'card', title: 'I am card'},
])
miro.board.tags.create({title: 'Red tag', color: '#F24726', widgetIds: widgets})

// Find all widgets with tag 'Red tag'
let tags = await miro.board.tags.get({title: 'Red tag'})
console.log(tags[0].widgetIds)
```

Listen to events on the board

To be notified of user interaction with the board, a web-plugin can create event listeners.

The Miro SDK supports the following events:

- SELECTION_UPDATED
- WIDGETS_CREATED
- WIDGETS_DELETED
- WIDGETS_TRANSFORMATION_UPDATED
- ALL_WIDGETS_LOADED

Example

```
miro.addListener('WIDGETS_CREATED', widget => {
  console.log(widget)
})

// For ALL_WIDGETS_LOADED event, we need to check if widgets
// are already loaded before subscription
async function onAllWidgetsLoaded(callback) {
  const areAllWidgetsLoaded = await miro.board.widgets.areAllWidgetsLoaded()
  if (areAllWidgetsLoaded) {
    callback()
  } else {
    miro.addListener('ALL_WIDGETS_LOADED', callback)
  }
}
onAllWidgetsLoaded(() => {
  console.log('all widgets are loaded')
})
```

Control the visibility of widgets for the current user

You can hide and show a widget for the current user by changing the `clientVisible` property of the widget. Widget visibility is applicable only in the user's web browser, and it does not affect other users of the board.

Example

```
// Locally hide all stickers for the current user
const allStickers = await miro.board.widgets.get({type: 'sticker'})
allStickers.forEach(s => {
        s.clientVisible = false
})
miro.board.widgets.update(allStickers)
```

Store metadata in a widget

You can store custom data in a widget. This data is public. It can be read by any other web-plugin, and also retrieved using the REST API.

Example

```
miro.board.widgets.create({
  "type": "sticker",
  "text": "some text",
  "metadata": {
    "{your_app_id}": {
      "hello": "world"
    }
  }
})
```

Disable editing on a widget

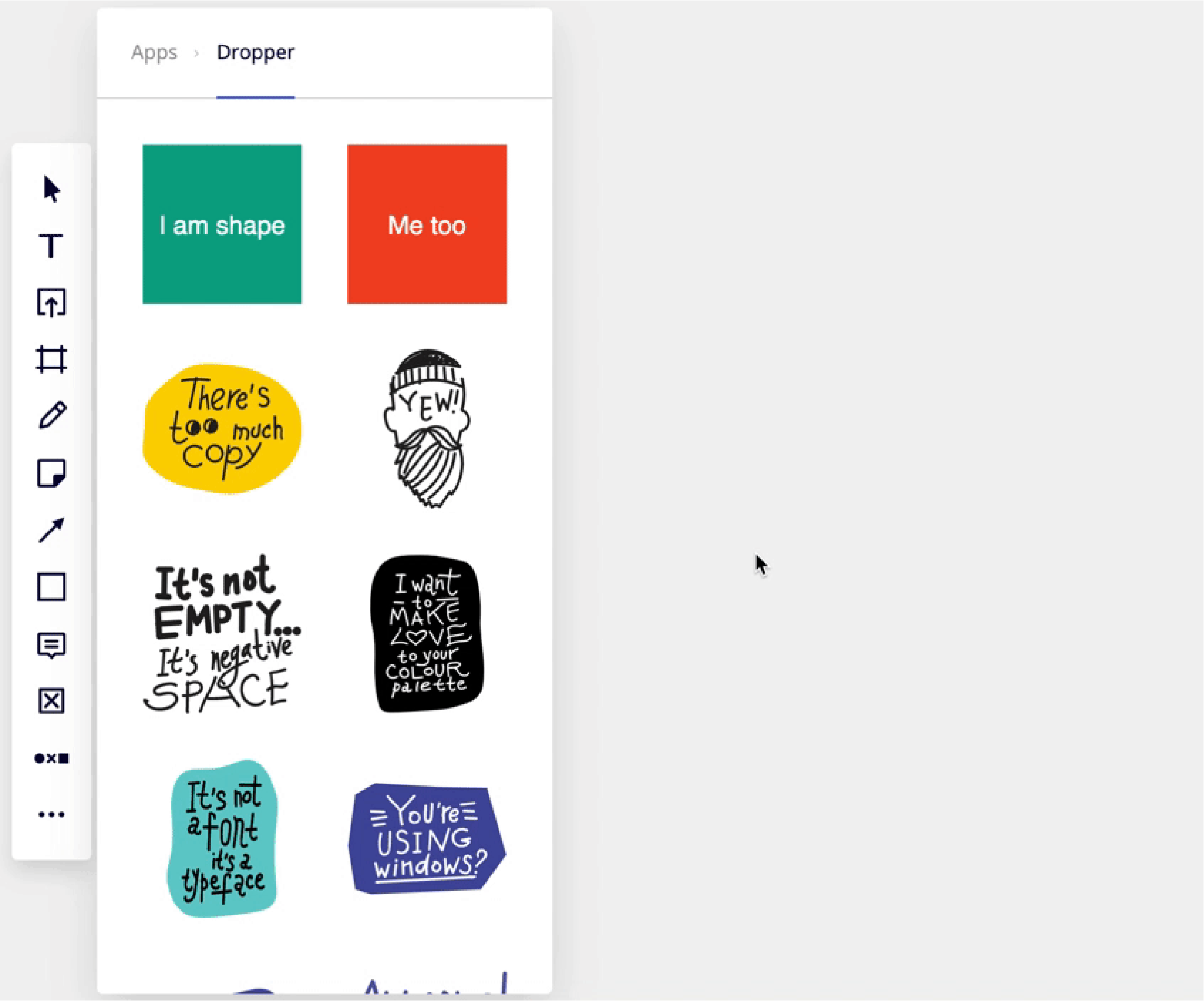
You can restrict all users from editing a widget by setting `widget.capabilities.editable=false`.

Example

```
// Create a sticker that is not editable
miro.board.widgets.create({
  "type": "sticker",
  "text": "some text",
  "capabilities": {
    "editable": false
  }
})
```

Enable drag and drop of items from a custom view

Custom views are often used to offer users a library of content they can add to a board, such as icons and images. You can enable drag and drop for such items using the `initDraggableItemsContainer()` method.


Apps › Dropper
I am shape
Me too
There's too much copy
YEW!
It's not EMPTY... It's negative SPACE
I want to MAKE LOVE to your COLOUR palette
It's not a font it's a typeface
You're USING windows?

Example

```
<body>
    <div id="box" style="background: red; width: 50px; height: 50px;"></div>
    <script>
    async function createWidget(canvasX, canvasY) {
        const widget = (await miro.board.widgets.create({type: 'shape', x:canvasX || 0, y:canvasY || 0}))[
        miro.board.viewport.zoomToObject(widget)
    }

    const options = {
      draggableItemSelector: '#box',
      onClick: async (targetElement) => {
        createWidget()
      },
      getDraggableItemPreview: (targetElement) => { //drag-started
        return {url: HOTSPOT_PREVIEW}
      },
      onDrop: (canvasX, canvasY) => {
        createWidget(canvasX, canvasY)
      }
    }

    miro.onReady(() => {
      miro.board.ui.initDraggableItemsContainer(document.body, options)
    })

    </script>
  </body>
```

Control the visible area of the board

A web-plugin can control the area of the board visible in the user's client. For example, zooming to a specific widget on the board.

Example

```
// Zoom to selected widget
let selectedWidgets = await miro.board.selection.get()
let sticker = selectedWidgets[0]
miro.board.viewport.zoomToObject(sticker.id)
```

All viewport methods

```
miro.board.viewport.getViewport(): Promise<IRect>
miro.board.viewport.setViewport(viewport: IRect): Promise<IRect>
miro.board.viewport.setViewportWithAnimation(viewport: IRect): Promise<IRect>

miro.board.viewport.zoomToObject(objectId: InputWidget, selectObject?: boolean)

miro.board.viewport.setZoom(value: number): Promise<number>
miro.board.viewport.getZoom(): Promise<number>
```

Manage widget selection

A web-plugin can create a listener on the `SELECTION_UPDATED` event to be notified of changes in the set of widgets that the current user has selected. The web-plugin can then query for the list of selected widgets using `miro.board.selection.get()`, and modify the selection using `miro.board.selection.selectWidgets()`.

Example

```
// Select all stickers on the board
let allStickers = await miro.board.widgets.get({type: 'sticker'})
miro.board.selection.selectWidgets(allStickers)
```

Display notifications

Web-plugins can display notifications to the user at the top of the board with these two methods:

- `showNotification(text: string)`
- `showErrorNotification(text: string)`

Add options to Miro settings

It is not currently possible for a web-plugin to add additional options to Miro settings.

Extension points

Miro apps are a combination of functionalities and entry points that are called **extension points**. As a developer, you can use these extension points to render your app in the product.

We provide two types of extension points:

1. **Button extension points**: allow a solution to be accessed from the canvas by adding one or more icon buttons in the UI:

- Toolbar
- Bottombar
- ExportMenu

2. **Container extension points**: allow rendering a custom HTML in a container on the canvas:

- Library
- Left Sidebar
- Bottom Panel
- Modal

> ### 🚧 An app needs an extension point
>
> Apps that are part of the UI will always use a button extension point. Some apps might require both button and container extension points to function.

How to choose between button and container extension points

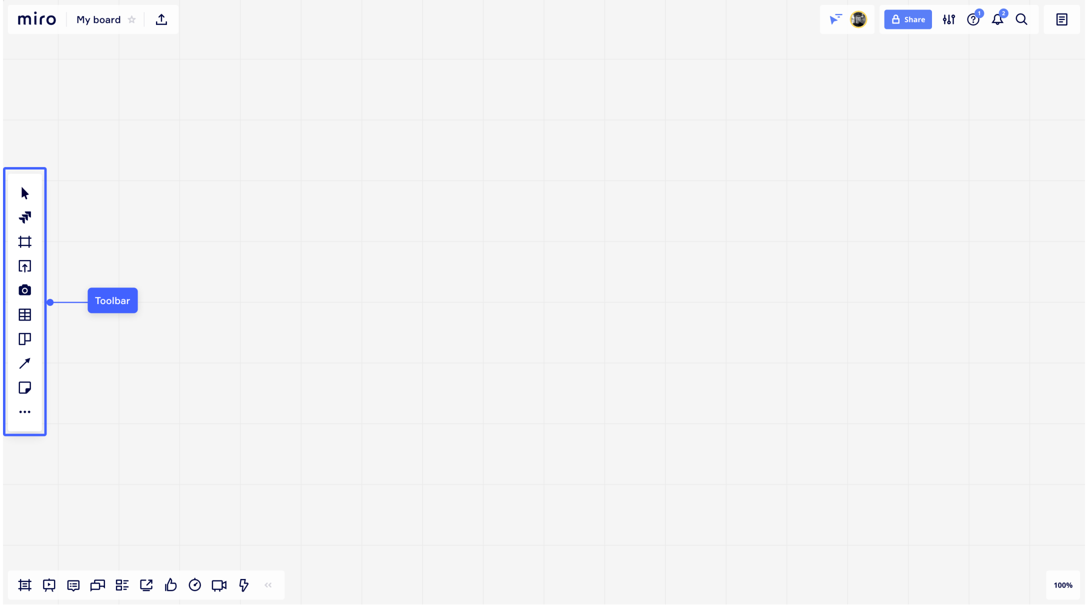
Some apps are simple and might require only a button extension point. For example, if you want to create an app that will duplicate the selection on your board, you don't need to use a container. You just need a button on the UI that triggers the action and modifies the board.

Other apps might require a combination of both the button and container extension points, first by accessing it via the button and then rendering the custom HTML in one of our container extension points. Unsplash, for example, is using a button extension point to trigger the Library container extension point to show images from Unsplash.

Button extension points

Toolbar

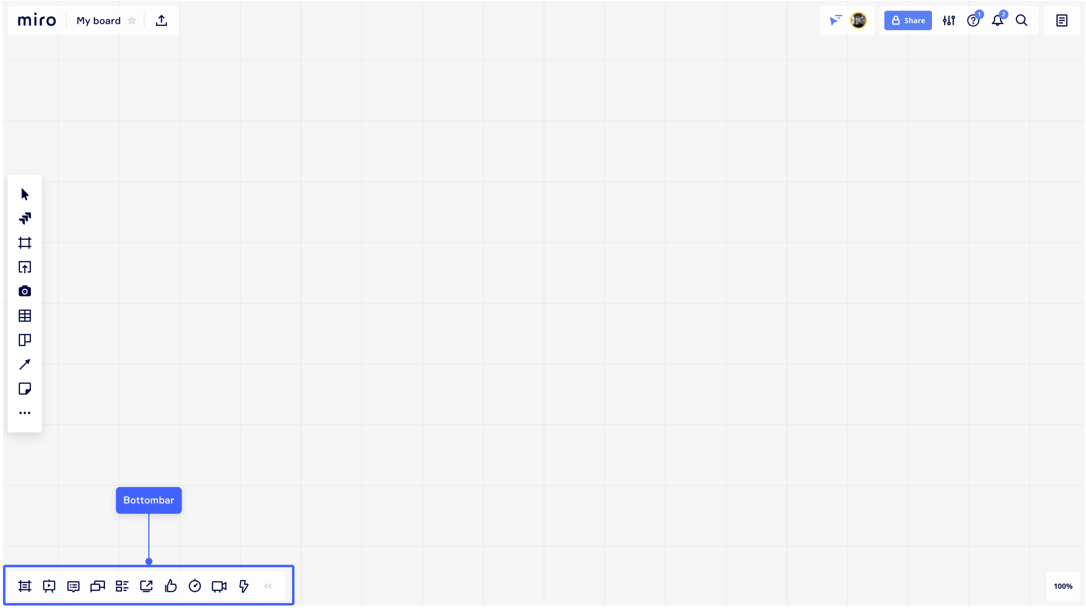
The toolbar button extension point is mainly used to render content apps for the canvas.



Apps will be present in the More icon at the bottom, but if you drag it you can make it to be visible all the time.

Bottombar

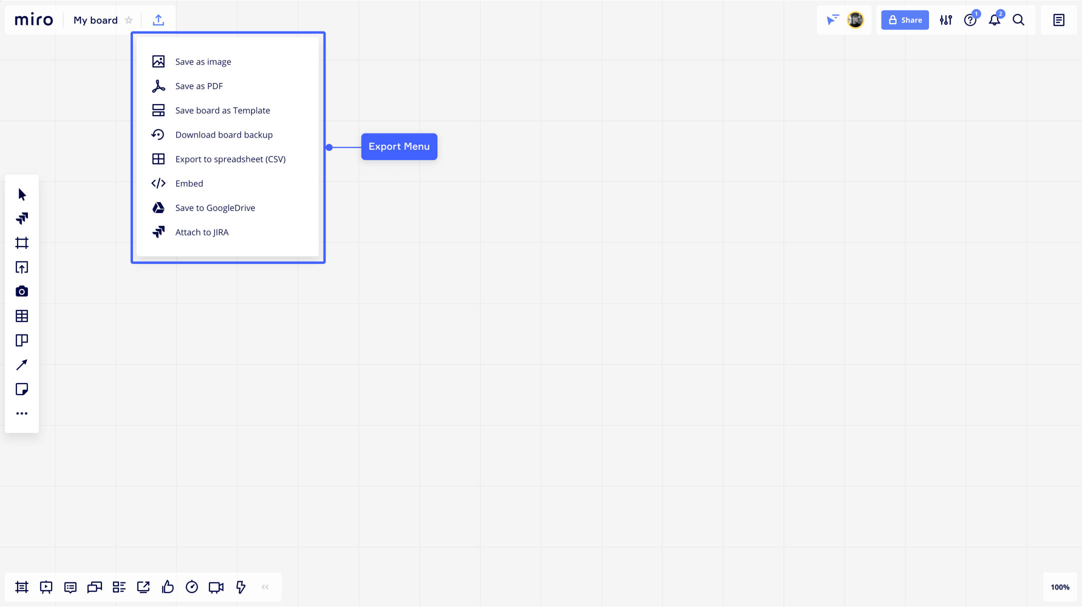
The bottom bar button extension point is mainly used to render collaboration apps or content that can be represented in a different format.



Comments is not an app that can be installed, but it is using the **bottomBar* extension point to present all comments which are on the board and display them in a different format.

Export Menu

The ExportMenu button extension point is used to render export functionalities.

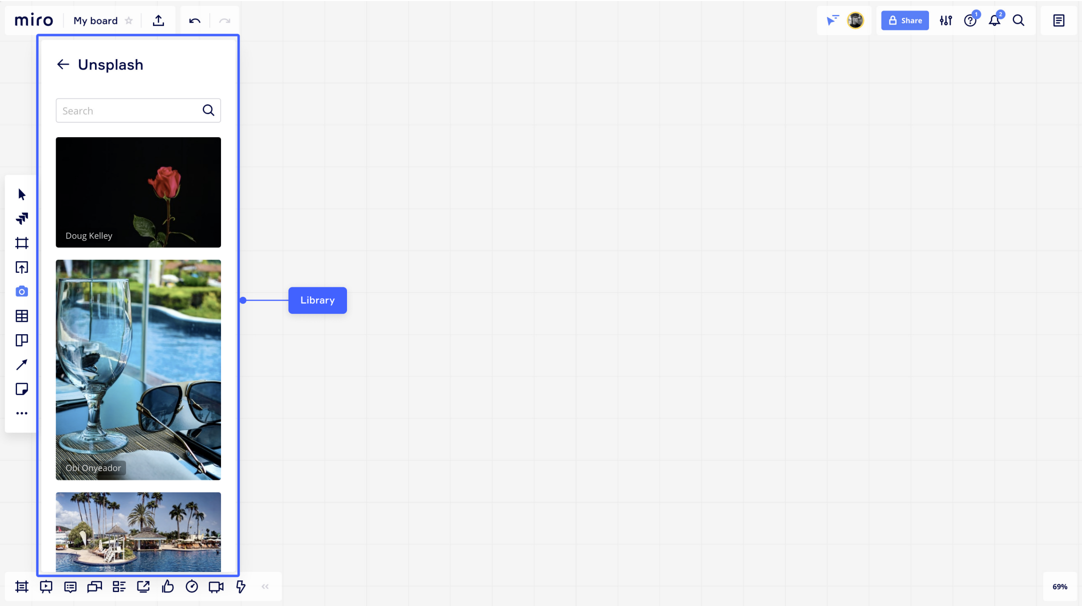


Google Drive is using the ExportMenu extension point to save and export your Miro board to Google Drive.

Container extension points

Library

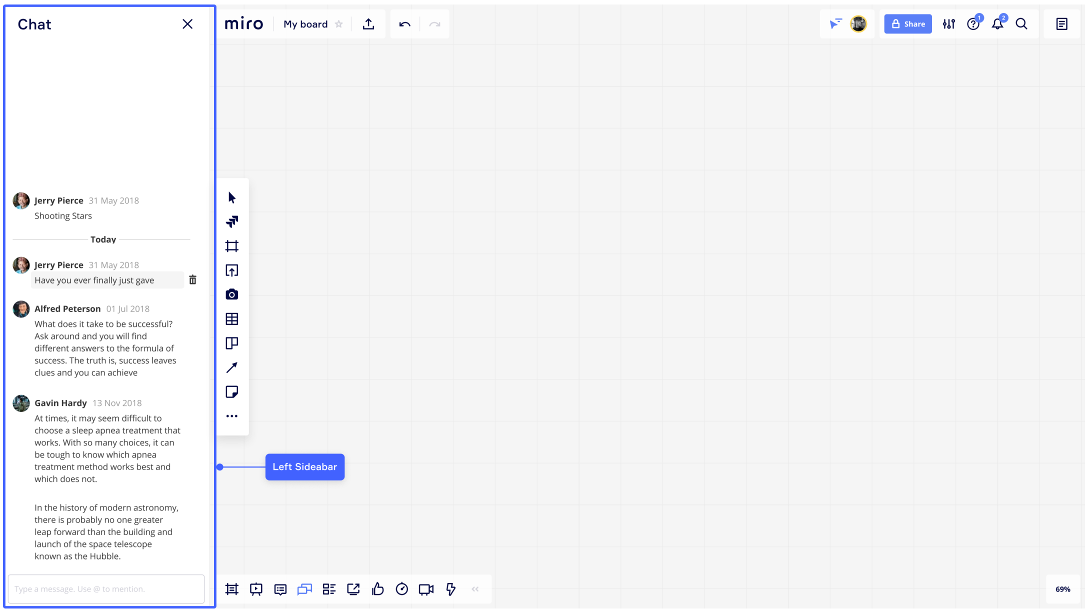
The Library is used to render materials that you can drag & drop on the canvas.



Example apps: Unsplash, IconFinder, AWS icon set

Left Sidebar

The Left Sidebar is used for collaborative apps, data search, automatization.



Chat

Jerry Pierce 31 May 2018
Shooting Stars

Today

Jerry Pierce 31 May 2018
Have you ever finally just gave

Alfred Peterson 01 Jul 2018
What does it take to be successful? Ask around and you will find different answers to the formula of success. The truth is, success leaves clues and you can achieve

Gavin Hardy 13 Nov 2018
At times, it may seem difficult to choose a sleep apnea treatment that works. With so many choices, it can be tough to know which apnea treatment method works best and which does not.

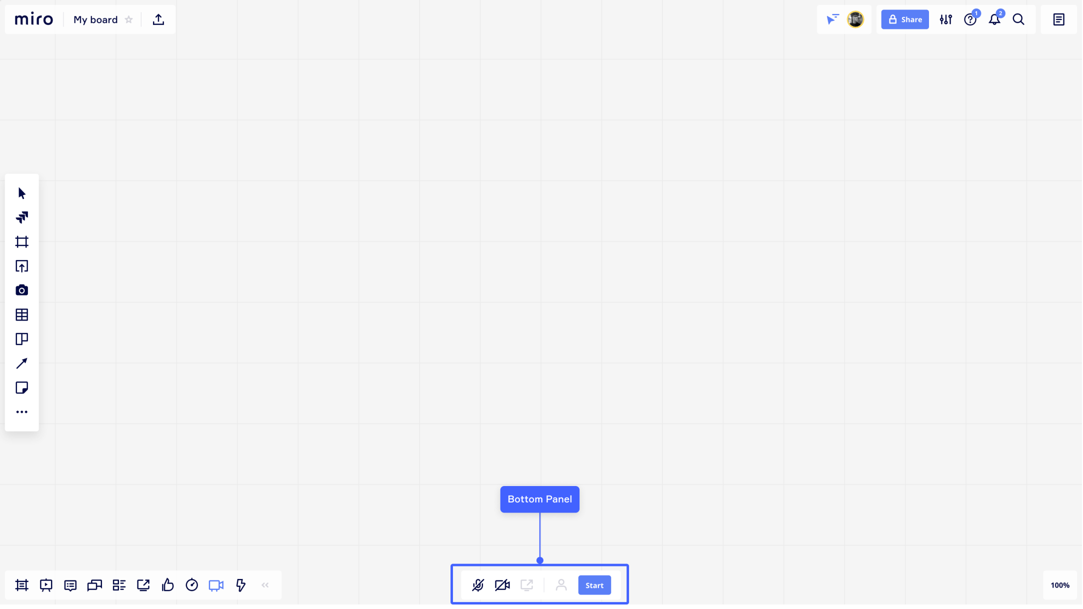
In the history of modern astronomy, there is probably no one greater leap forward than the building and launch of the space telescope known as the Hubble.

Type a message. Use @ to mention.

Example apps: Chat

Bottom Panel

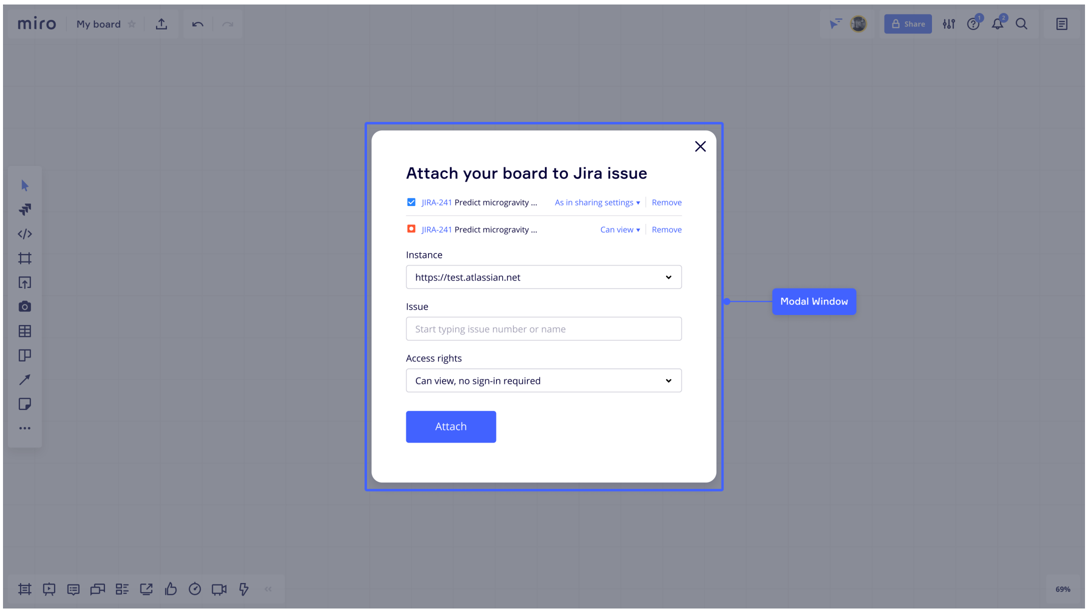
The Bottom Panel, is used for apps that require interactions while still keeping the board available. It can be extended to 320px width and 200px height.



Example apps: Video chat, Timer

Modal Window

The Modal window, is used for dialogs that require a decision to be made or to be filled with content.



Attach your board to Jira issue

☑ JIRA-241 Predict microgravity ... As in sharing settings ▾ | Remove

JIRA-241 Predict microgravity ... Can view ▾ | Remove

Instance

https://test.atlassian.net

Issue

Start typing issue number or name

Access rights

Can view, no sign-in required

Example apps: Jira

How to use extension points

Extension point — is a place in product where web-plugins can render user interface.

Miro provides two types of extension points:

- **Custom view** to render custom HTML in a container;
- **Button extension points** to add a button.

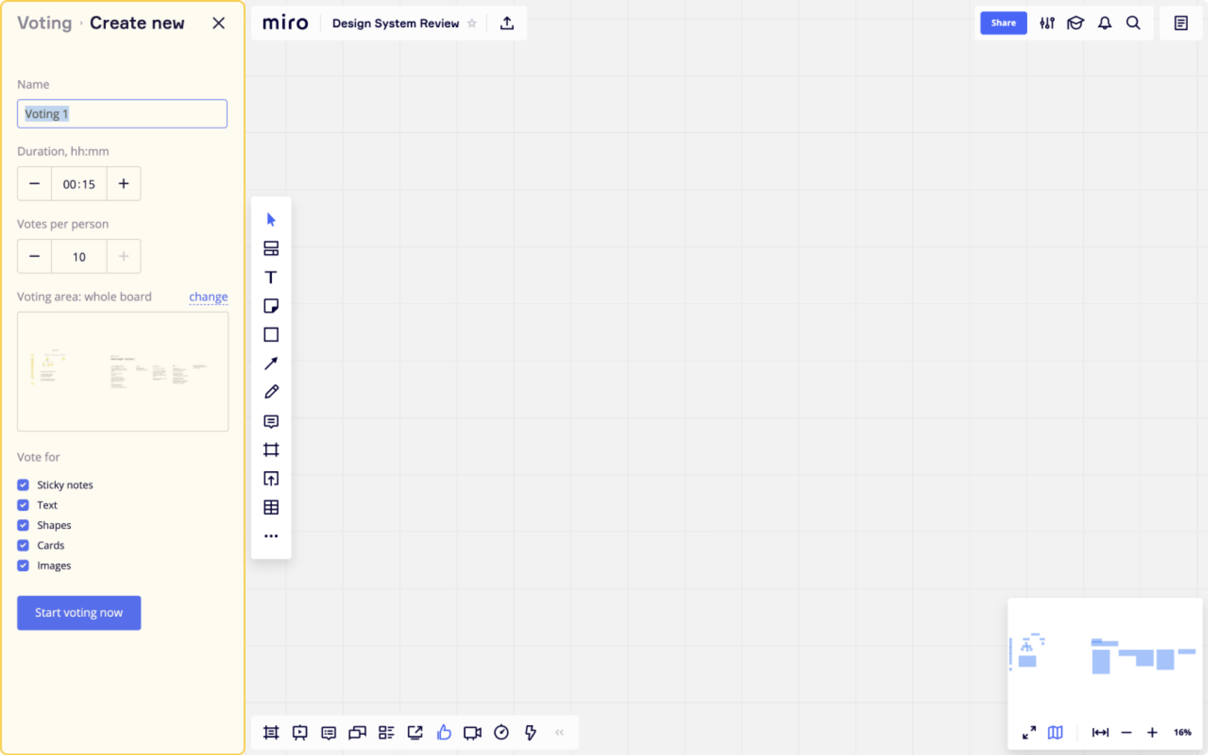
Custom view:

- LeftSidebar;
- Library;
- BottomPanel;
- ModalWindow.

Button extension points:

- toolbar;
- bottomBar;
- exportMenu;
- widgetContextMenu.

Custom view. LeftSidebar.



Voting · Create new

Name: Voting 1

Duration, hh:mm: 00:15

Votes per person: 10

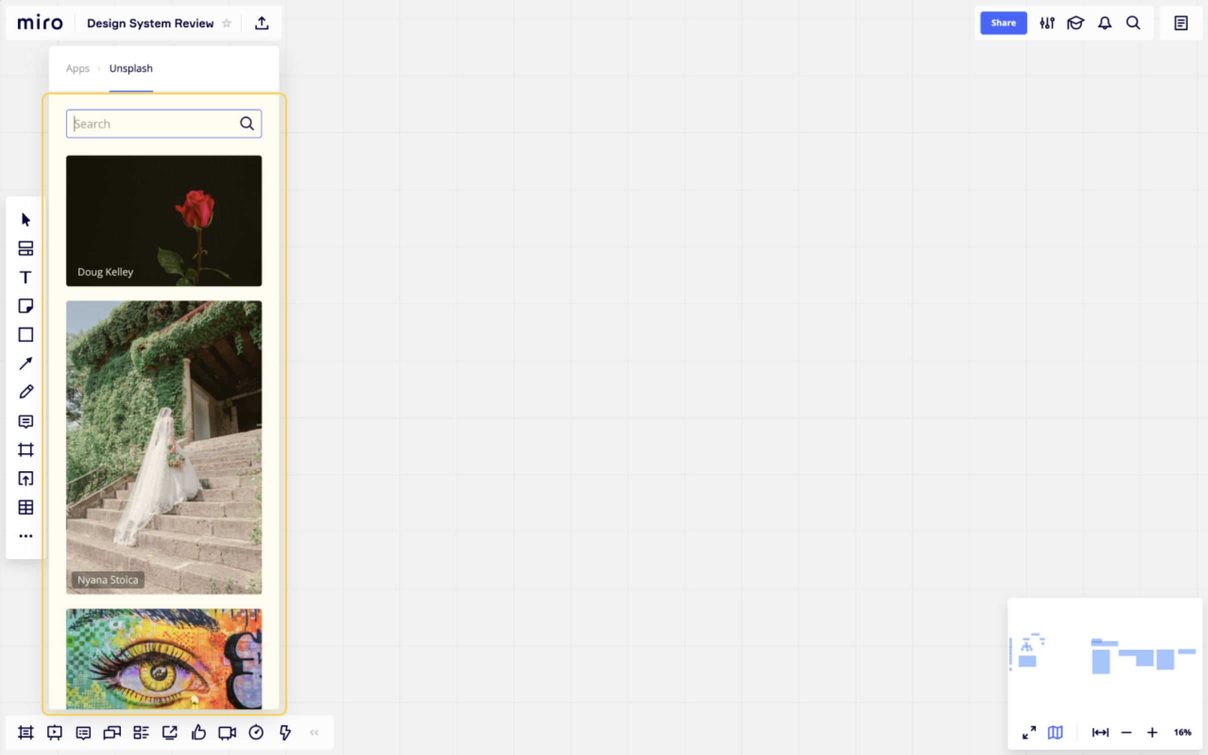
Voting area: whole board change

Vote for

☑ Sticky notes
☑ Text
☑ Shapes
☑ Cards
☑ Images

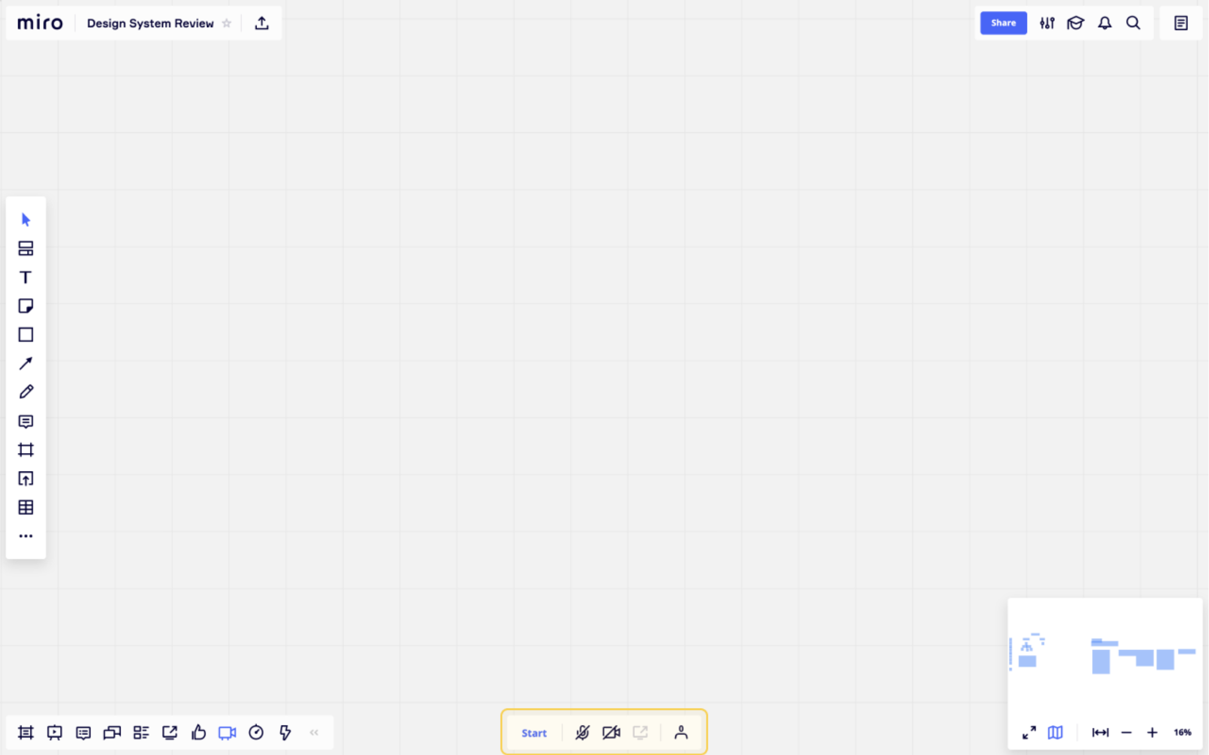
Use it for collaborative plugins, data search, automatization. Don’t use it for adding widgets to canvas. LeftSidebar fixed width = 340px.

Custom view. Library.



The library contains materials that you can drag & drop to the canvas.
Don’t use the library for other use-cases.

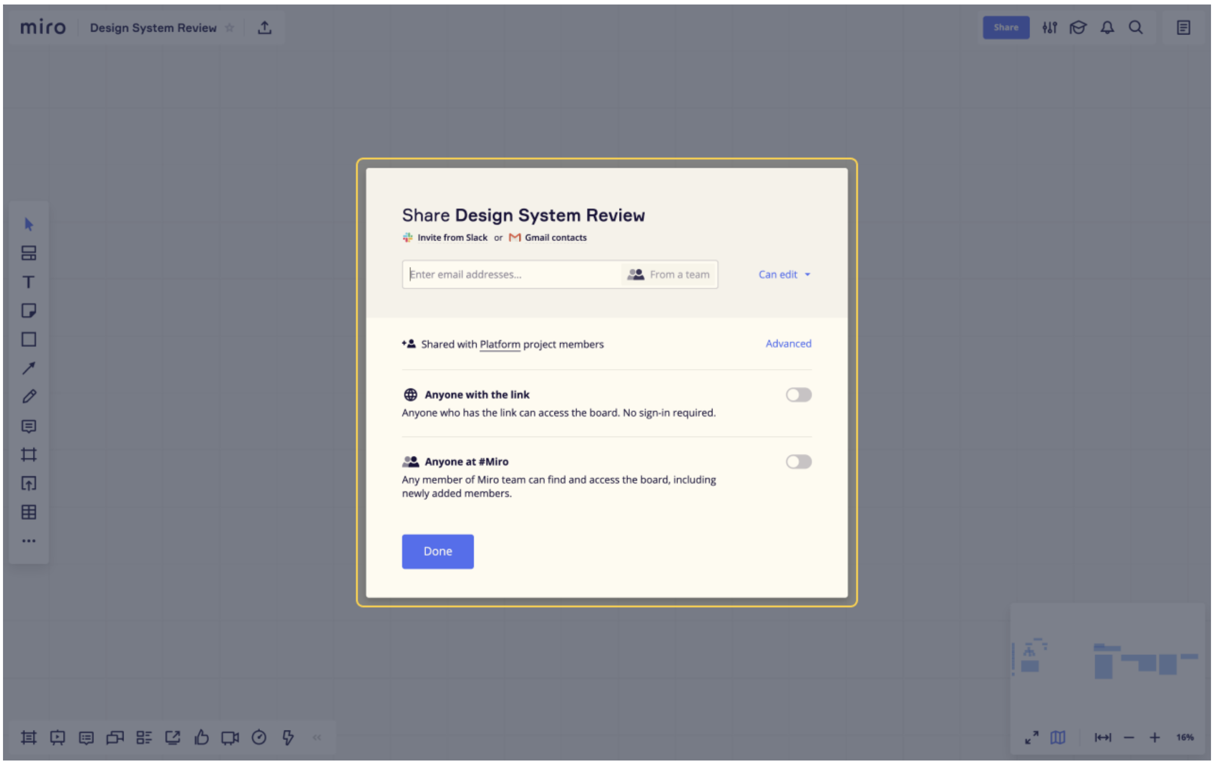
Custom view. BottomPanel.



You can add up to 5 simple controls to the BottomPanel that don't take too much space and are needed right in action. Don’t use it for alerts.

You can change BottomPanel width up to 320px and height up to 200px.

Custom view. ModalWindow.



Share Design System Review

Invite from Slack or Gmail contacts

Enter email addresses... From a team Can edit

Shared with Platform project members Advanced

Anyone with the link

Anyone who has the link can access the board. No sign-in required.

Anyone at #Miro

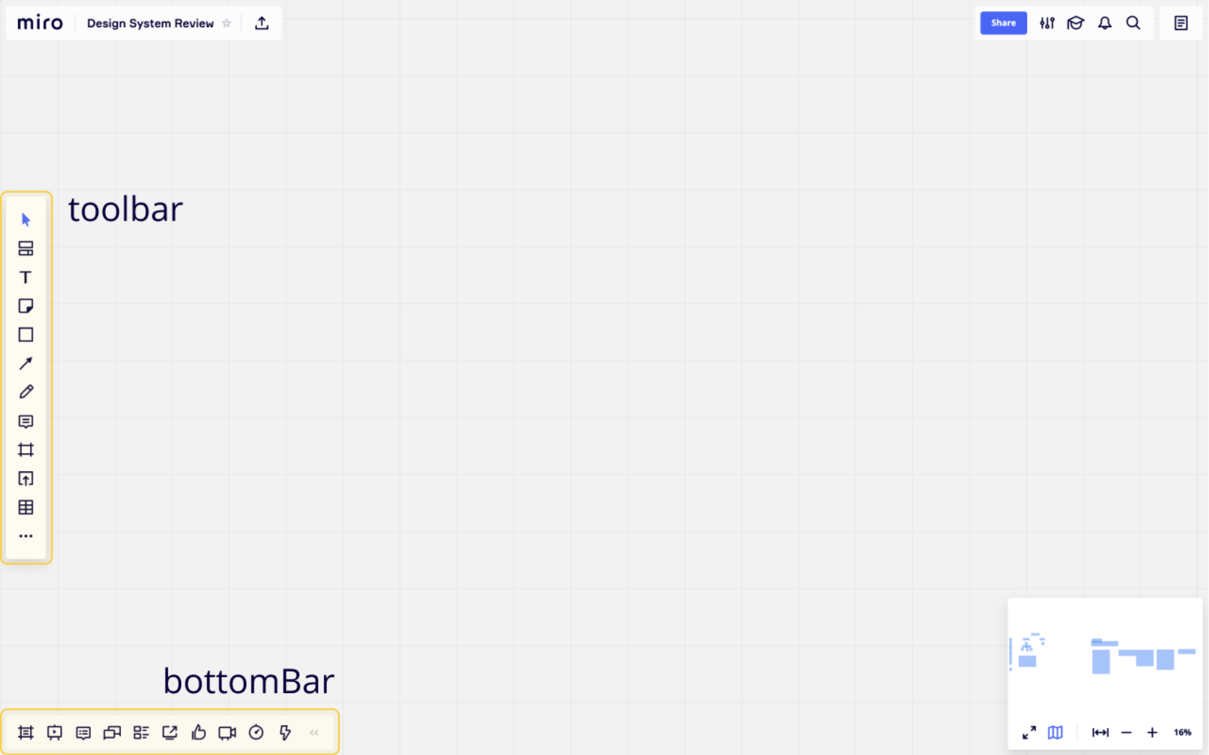
Any member of Miro team can find and access the board, including newly added members.

Done

Use it for dialogs, that require user decision.
Or fill out modal window with content, for example: jira tasks picker.

ModalWindow has max width and height = 80%.

Button extension points. toolbar and bottomBar.

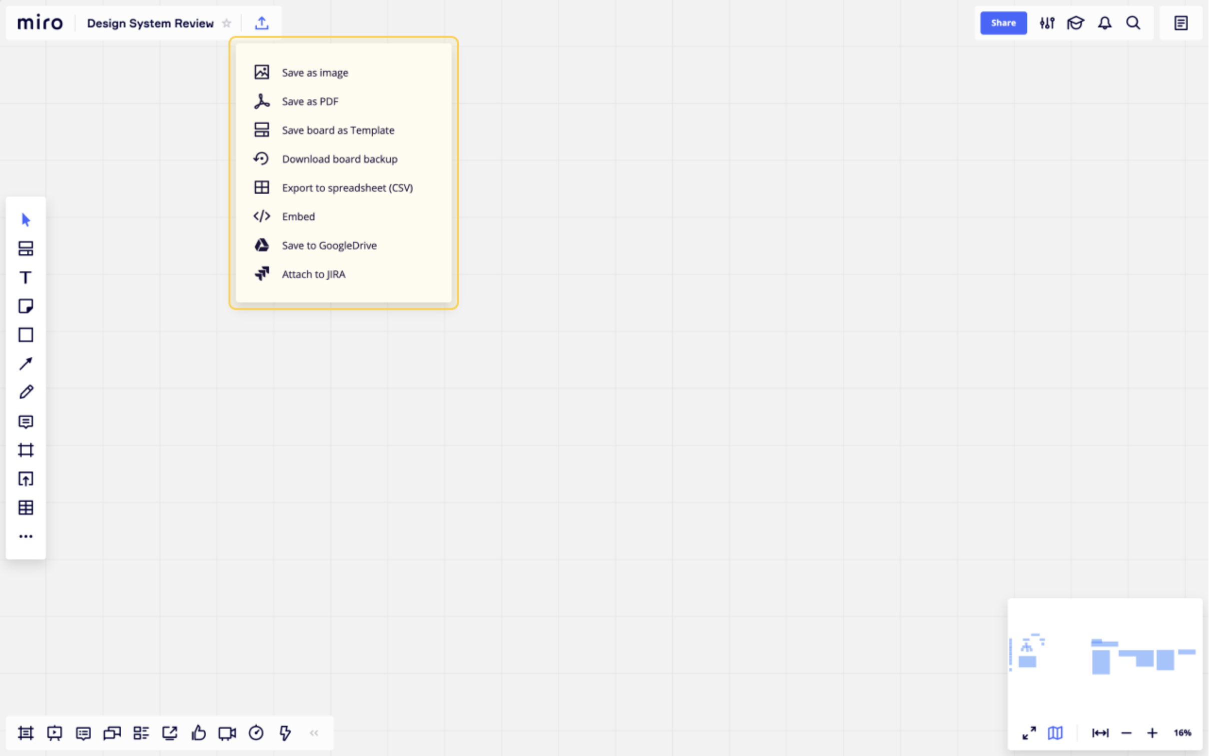


toolbar contains content sources for canvas, for example: icon library, wireframing library, templates, etc.

bottomBar contains either collaboration plugins, like chat, or another way to represent content, like presentation mode.

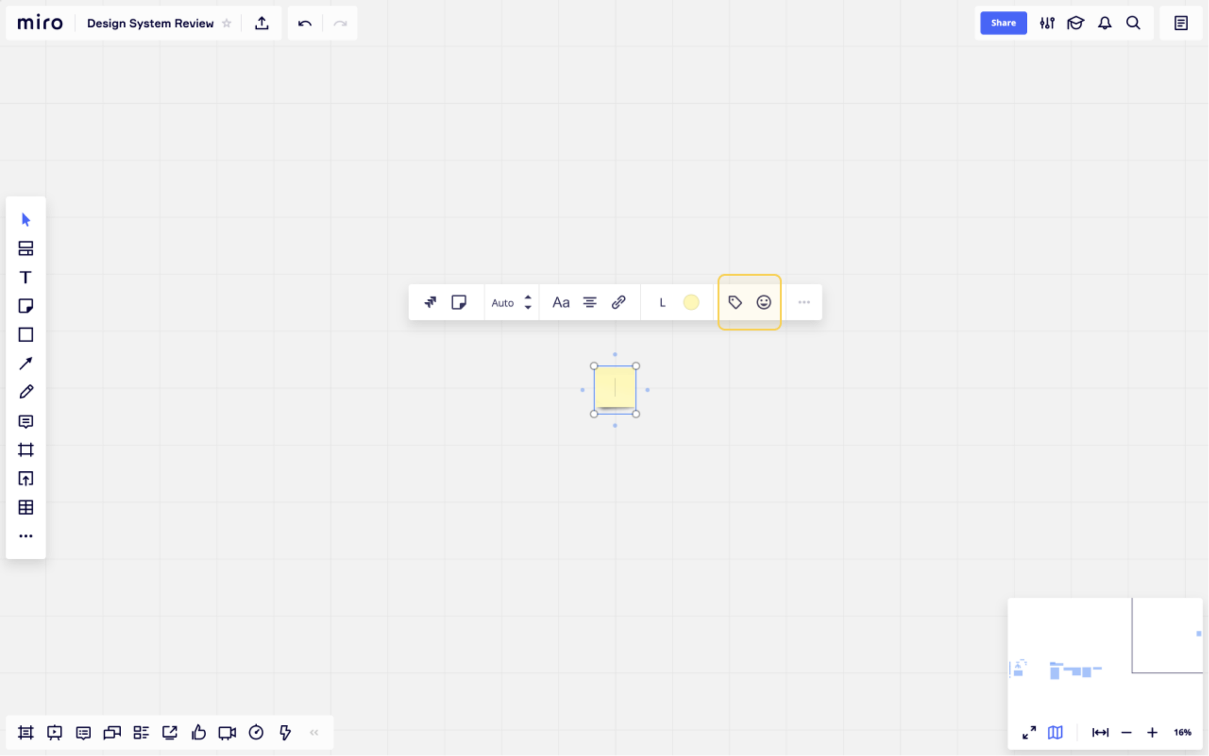
Don't use them for other use-cases.

Button extension point. exportMenu.



Use it for export options.
Don't use it for other use-cases.

Button extension point. widgetContextMenu.



All buttons will be added to the right side of the toolbar. You can only choose the number of the buttons, but can not change their location.

Use it to add options that affect selected element(s).
Don’t use it for options, that affect unselected elements.

Web-plugin Authorization

After your web-plugin is installed, it is available in the team and to all team members. Before the web-plugin can access any board content or open windows, each team member must first authorize the web-plugin and grant the required permissions. To check and request authorization, you'll need two methods: `miro.isAuthorized` and `miro.requestAuthorization`. Here's an example of how to use these methods:

JavaScript

```javascript
async function onToolbarAppButtonClicked () {
  const isAuthorized = await miro.isAuthorized()

  if (!isAuthorized) {
    // Ask the user to authorize the app.
    await miro.requestAuthorization()
  }

  // Once authorized, open the app.
  openMyApp()
}

function openMyApp () {
  miro.board.ui.openLeftSidebar('main-app-sidebar.html')
}
```

Authorization of apps using the Miro REST API

If your app uses both the client SDK and the Miro REST API, you'll need to provide extra parameters to `miro.requestAuthorization` to ensure that the user's access credentials are passed to your application backend for storage and future use:

JavaScript

```
async function onToolbarAppButtonClicked () {
  const isAuthorized = await miro.isAuthorized()
  if (isAuthorized) {
    // Open the app.
    openMyApp()
  } else {
    // State is an optional parameter.
    // You can use it to pass extra parameters to the redirect URI page or for security purposes.
    // See https://developers.google.com/identity/protocols/oauth2/web-server#creatingclient,
    const installState = await fetch(
      'https://my-app-server.com/api/install-state',
      { credentials: 'same-origin' }
    ).then(res => res.text())

    // Ask the user to authorize the app.
    await miro.requestAuthorization({
      // To successfully complete the miro.requestAuthorization call,
      // the redirect URI page must redirect the user back to Miro.
      // See the documentation below.
      redirect_uri: 'https://my-app-server.com/install',
      state: installState
    })

    // Once authorized, open the app.
    openMyApp()
  }
}

function openMyApp () {
  miro.board.ui.openLeftSidebar('main-app-sidebar.html')
}
```

Once the app's backend has successfully completed the authorization flow for the Miro REST API, the redirect URI page must redirect the user back to the following Miro page:

HTTP

```
https://miro.com/app-install-completed/?client_id=YOUR_APPLICATION_ID&team_id=ID_OF_THE_TEAM_THE_APP_HAS_B
```

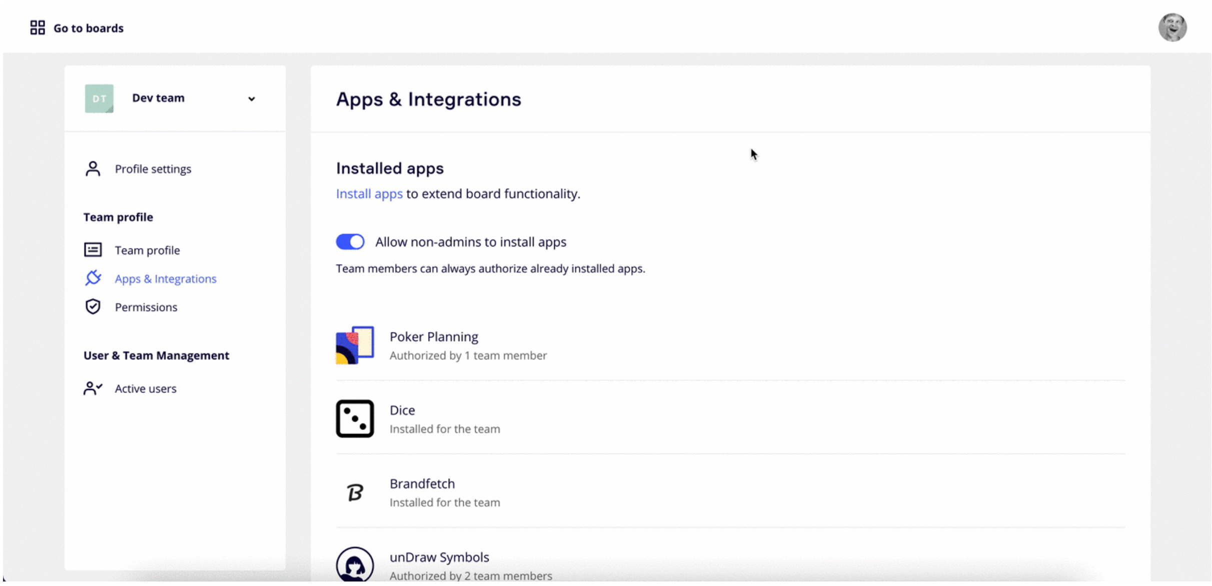
Where:
The `client_id` is the Client id on the Miro App Settings page. App dashboard →
The `team_id` is the team_id from the JSON response containing the non-expiring access_token. For more information, see Getting an access token.

Testing the authorization for your web-plugin during development

To verify the authorization for your web-plugin:

1. Go to the team app settings.

2. Find your app and uninstall the app only for yourself (not for the whole team).



Apps & Integrations

Installed apps

Install apps to extend board functionality.

Allow non-admins to install apps

Team members can always authorize already installed apps.

Poker Planning
Authorized by 1 team member

Dice
Installed for the team

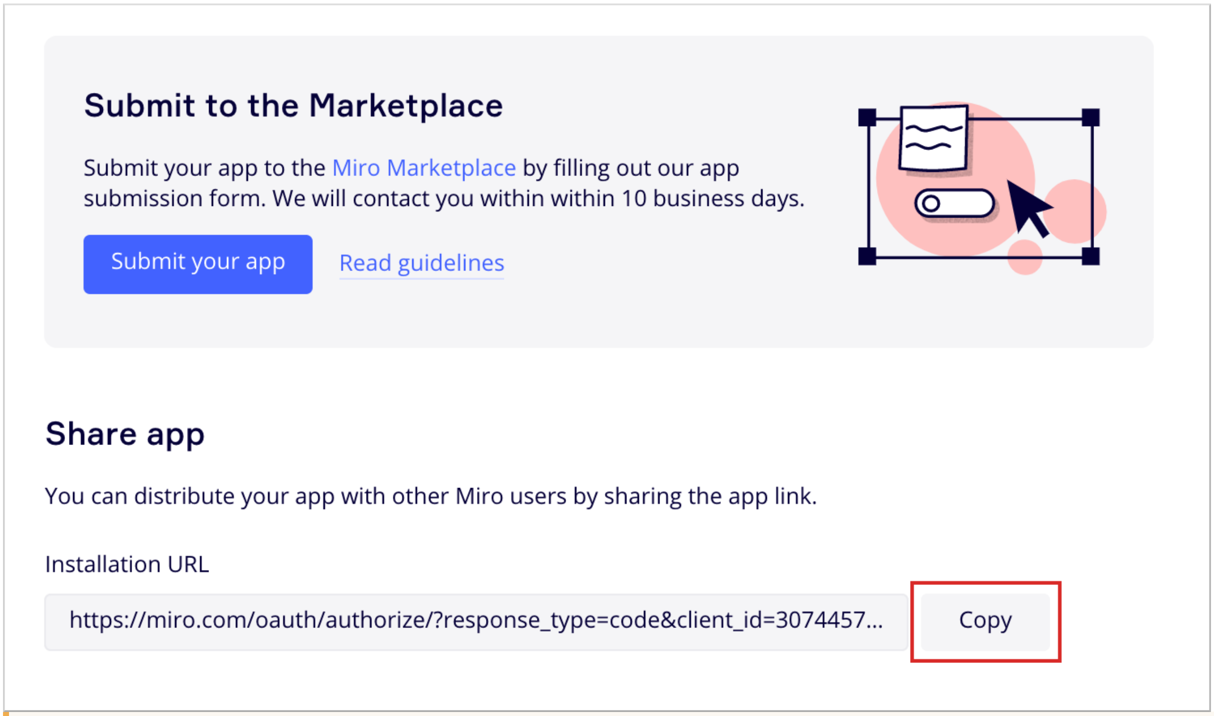
Brandfetch
Installed for the team

unDraw Symbols
Authorized by 2 team members

3. Open a board within the same team.
4. Use your app to verify that the app checks and requests authorization.

Share your app

You can share your app with other Miro users, by sharing the installation link. You can find the installation link for each app when you scroll down to the **Share app** section on your app's settings page. [App dashboard →]()



Submit to the Marketplace

Submit your app to the Miro Marketplace by filling out our app submission form. We will contact you within within 10 business days.

Share app

You can distribute your app with other Miro users by sharing the app link.

Installation URL

https://miro.com/oauth/authorize/?response_type=code&client_id=3074457...

> 🚧 Note:
>
> After the first installation, your web-plugin is installed in the user's team and is available to all team members. If your web-plugin requires user permission to be granted for OAuth scopes, such as *boards:read* or *boards:write*, each team member needs to authorize the web-plugin.

FAQs

How do I position a widget?

You position a widget by modifying the `x` and `y` properties of the widget, which are the coordinates of the center of the widget.

Is it possible to display buttons in the BottomBar only for certain users?

You can configure buttons to be displayed only to selected users, including:

- authorized users
- editors
- team members
- logged in users

A code sample illustrating this is available at:
https://github.com/miroapp/app-examples/tree/v1//buttons

How can I give a button a custom icon?

Button icons are defined in SVG format as shown below. To support the blue hover effect, use `fill="currentColor"`.

JavaScript

```
miro.onReady(run)

const icon24 =
  '<path fill="currentColor" fill-rule="nonzero" d="M20.156 7.762c-1.351-3.746-4.672-5.297-8.838-4.61-3.9.

const icon48 =
  '<path fill="#5B00FF" fill-rule="nonzero" d="M2.5 16.5C6 1.5 37-8 44.67 13.694c5.8 16.41-1.206 27.407-20

function run() {
  miro.initialize({
    extensionPoints: {
      toolbar: {
        title: 'Template builder',
        toolbarSvgIcon: icon24,
        librarySvgIcon: icon48,
        onClick: () => {
          miro.board.ui.openLibrary('library.html', {title: 'Template builder'})
        },
      },
    },
  })
}
```

How do I make content added to a sidebar scrollable?

This can be done using the CSS `overflow` property, as shown below:

HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }

        .scrollable-container {
            height: 100%;
            overflow-y: auto;
        }

        .scrollable-content {
            height: 2000px;
            background-color: #2a79ff;
        }
    </style>
</head>
<body>
<div class="scrollable-container">
    <div class="scrollable-content"></div>
    Bottom
</div>
</body>
</html>
```

How do I position content in the center of the user's current view?

You can obtain the location and dimensions of the user's current view using `miro.board.viewport.get()` and position widgets accordingly. Remember that the positioning of widgets is based on their center point, so you will need to calculate the center of the current viewport as shown below.

JavaScript

```javascript
const viewport = await miro.board.viewport.get()
const centeredX = viewport.x + viewport.width / 2
const centeredY = viewport.y + viewport.height / 2
miro.board.widgets.create({type:'sticker', x:centeredX, y:centeredY})
```

Embed Miro Whiteboard

Overview

This documentation section explains how to embed a live collaborative Miro board into your website through an iframe. You can embed a Miro board into any web application, product, or website free of cost.

Approaches and APIs to embed a Miro board

The following sections describe different ways to embed a Miro board.

1. Live Embed via direct board link

If you know the board ID, you can embed an existing board by creating a direct embed link.

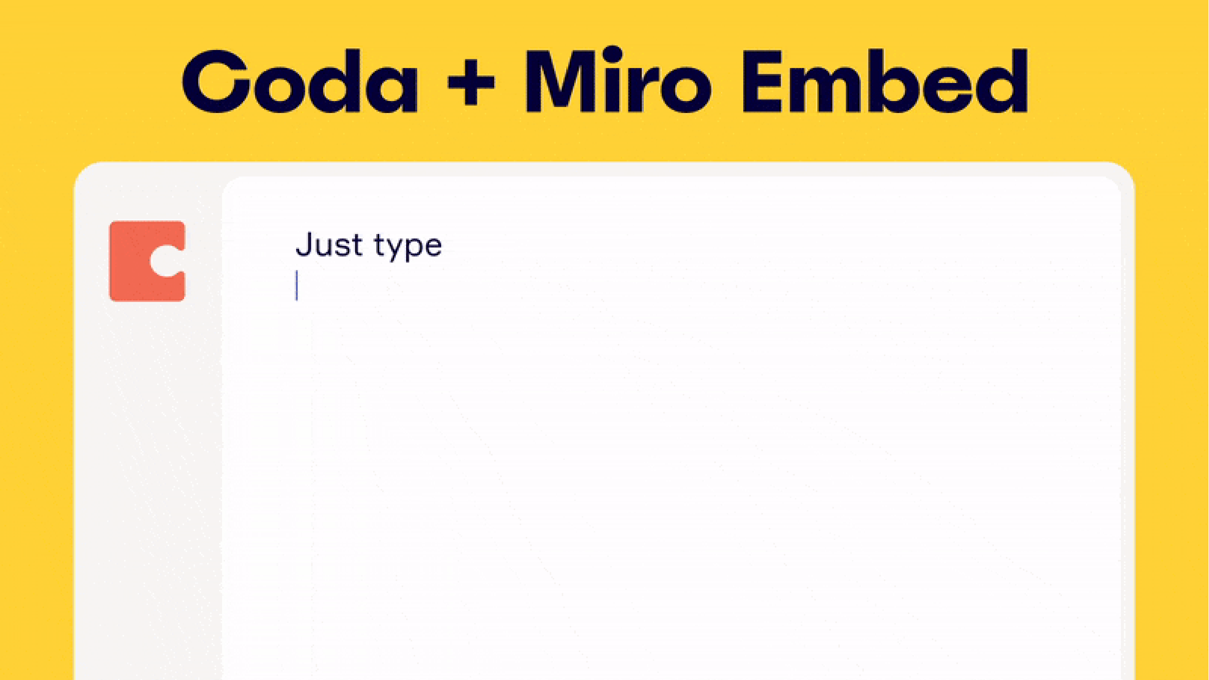
Use the following URL:

Embed board by direct link

```
https://miro.com/app/live-embed/{board_id}
```

The board is shared with its current sharing settings.

2. Using Boards Picker from JavaScript



This approach works as follows:

- Implement the Boards Picker component that allows the user to log on to Miro, pick a board, and choose the sharing settings with which the user wants to embed the board.
- In response, the Boards Picker component provides you with the board_id and access_link for this board, which you can use to embed the board that was picked.

See an example flow in Coda + Miro integration.

⚠️ The Boards Picker component requires enablement on Miro side ⚠️
Read more on how to set up Boards Picker.

3. Editable temporary boards for unregistered users

With this approach, the user of your integration can embed a temporary Miro board without being a registered Miro user.

See an example flow in Whereby + Miro integration.

Features:

1. `access-link` — allows users to get a special access-link for the selected board and embed it on the website via iframe.
2. `allowCreateAnonymousBoards` you can allow not registered in Miro users to create a new board and embed it on the website via iframe.

⚠️ this flow requires enablement on Miro side ⚠️
Read more how to set up editable boards for anonymous users

Embed a board via direct link

Getting started

To embed a Miro board, use the following URL:
`https://miro.com/app/live-embed/{board_id}`

Example: a Miro board embedded into an iframe via a direct link:

Here is the code that created this example:

Direct link embed iframe code example

```
<iframe width="768" height="432" src="https://miro.com/app/live-embed/o9J_kkQxX78=/?moveToViewport=-23165,
```

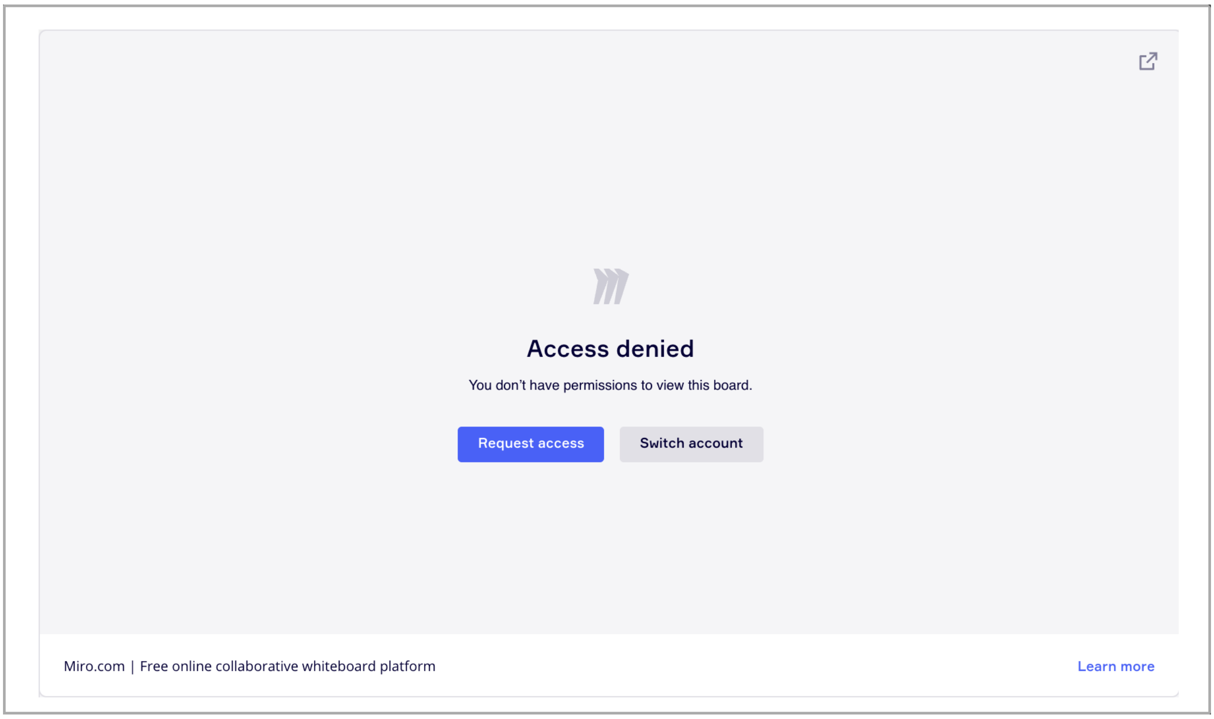
Get the board_id

To obtain the `board_id` manually, open the Miro board in your browser. Copy the ID from the browser URL: `https://miro.com/app/board/{board_id}/`

If you prefer a programmatic way to obtain the ID for a board, you can use the Miro REST API or the Boards Picker visual component.

Board access rights

Embedding a Miro board via a direct link uses the same access rights as the board itself, accounting for both user logins and board accessibility by link. This means that if a user can access the Miro board, the user can also access the embedded board. However, if the user does not have the required permission to access the board, the user is also unable to see the embedded board’s content (figure 1).



Access denied

You don't have permissions to view this board.

Miro.com | Free online collaborative whiteboard platform

Figure 1. *Access denied* error, shown when a user does not have the required permissions to view the board.

Note that access issues can also occur if the user's browser blocks third-party cookies.

You can use the Boards Picker component to grant users additional permissions for an embedded board by using an `access-link`. For example, a user who normally has only View rights to a board can be granted Edit rights through the embedded board. For more information on the boardsPicker component, see the documentation.

Optional link parameters

There are a few parameters that can be added to a direct link to customize the initial appearance of an embedded Miro board.

Optional link parameter	Description	Possible values
`embedAutoplay`	Indicates whether to display or skip the preloader screen.	`true` : Skips the preloader screen `false` : Loads the preloader screen Default: `false`
`moveToWidget`	Sets the start view of the board's content to the specified widget.	`widget_id` : The ID of the widget that must appear in the start view
`moveToViewport`	Sets the start view of the board's content to the specified viewport.	`x` : The x-coordinate for the upper-left corner of the viewport `y` : The y-coordinate for the upper-left corner of the viewport `width` : The minimum width of the viewport `height` : The minimum height of the viewport
`boardsAccessToken`	Provides an access token for user authentication on the embedded board, when using the Boards Picker component.	This parameter is added by the Boards Picker component, and cannot be set manually.

Skip the *See the board* screen

By default, Miro Live Embed opens with a preloader screen that displays the name of the board. The user must click a button on this screen to load the board (figure 2).

If you prefer to skip the preloader screen and immediately display the content of the board when the web page loads, you can set the `embedAutoplay` parameter to `true` .

`https://miro.com/app/live-embed/{board_id}/?embedAutoplay=true`

> Note: Any board embedded via the Boards Picker component will automatically include the embedAutoplay=true parameter, skipping the preloader screen.

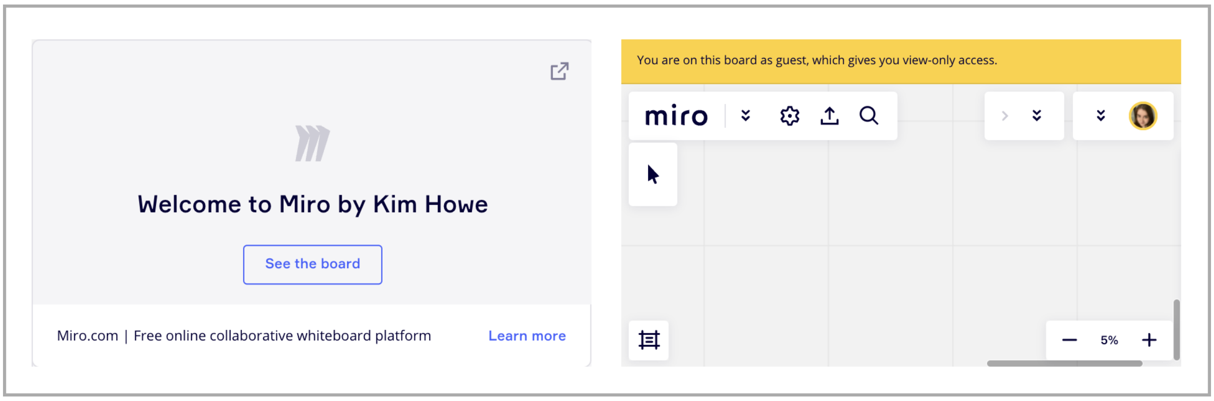

Welcome to Miro by Kim Howe
See the board
Miro.com | Free online collaborative whiteboard platform
Learn more
You are on this board as guest, which gives you view-only access.
miro
5%

Figure 2. The image on the left shows an example of the preloader screen. The image on the right is an example of setting `embedAutoplay=true` and shows the contents of the Miro board.

Set the start view

You can load an embedded Miro board with a specific start view, displaying either a specific widget (figure 3) or a specific viewport (figure 4).

> 🚧 Note:
>
> You **cannot** use both the `moveToWidget` and the `moveToViewport` parameters in a single embed link.

Display a specific widget as the start view

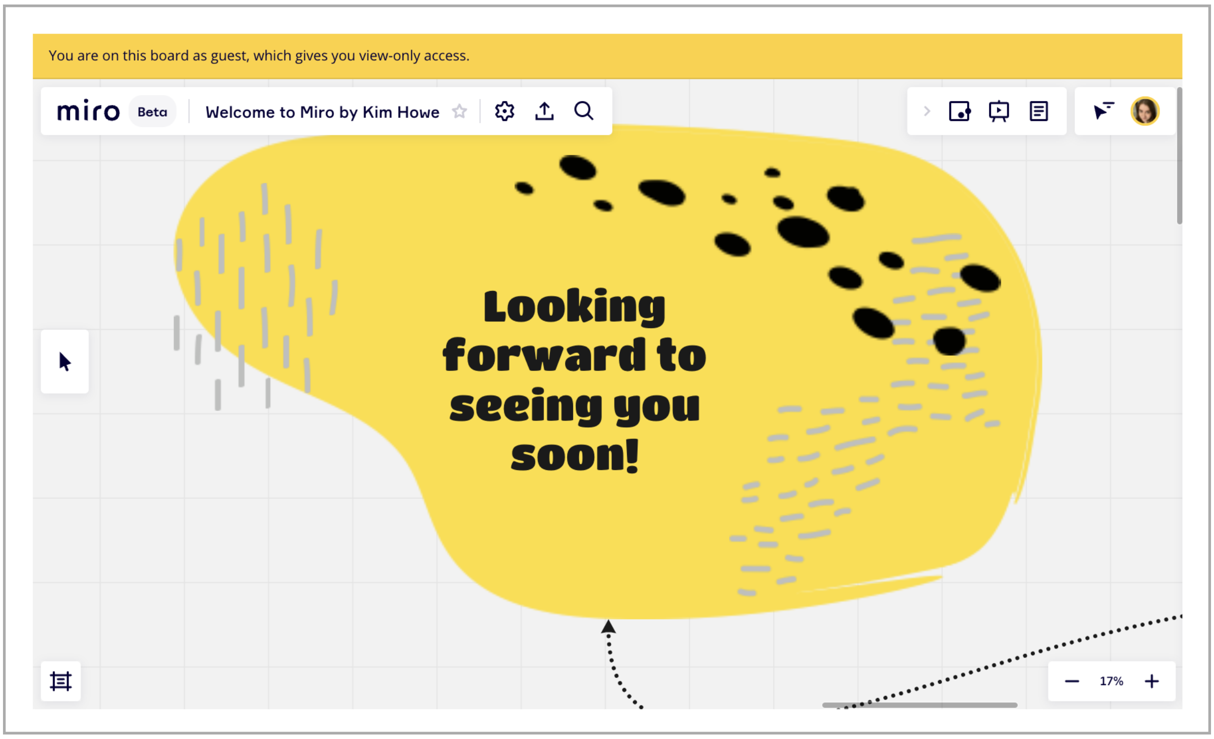
To display a specific widget as the start view for an embedded board, include the `moveToWidget` parameter in the direct link:

```
https://miro.com/app/live-embed/{board_id}/?moveToWidget={widget_id}
```

You can get a widget's ID programmatically or manually. To get the widget ID programmatically, use the Miro REST API.

To get the widget ID manually:

1. Locate the widget on the Miro board.
2. Right-click on the widget, and select **Copy link**.
3. Identify the widget ID from the link that you copied. The link structures mirror each other, so you should be able to find the widget ID attached to the `moveToWidget` parameter.
   ```
   https://miro.com/app/board/{board_id}/?moveToWidget={widget_id}
   ```



You are on this board as guest, which gives you view-only access.

miro
Beta
Welcome to Miro by Kim Howe
Looking forward to seeing you soon!
17%

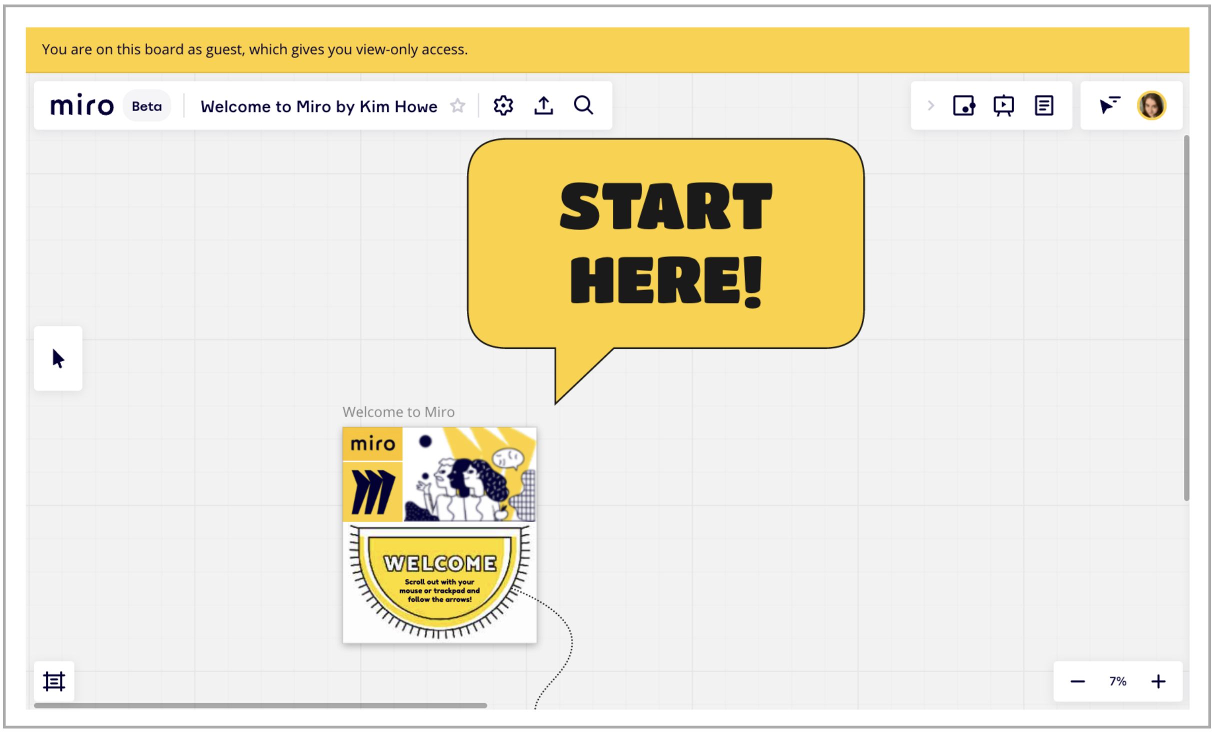
Figure 3. Example of an embedded Miro board with a start view that includes a specified widget on that board.

Display a specific viewport as the start view

To display a specific viewport as the start view for an embedded board, include the `moveToViewport` parameter in the direct link:

```
https://miro.com/app/live-embed/{board_id}/?moveToViewport={x},{y},{width},{height}
```

The only way to get the current viewport is by using the Web-Plugins API.


You are on this board as guest, which gives you view-only access.

miro
Beta
Welcome to Miro by Kim Howe
START HERE!
Welcome to Miro
miro
WELCOME
Scroll out with your mouse or trackpad and follow the arrows!
7%

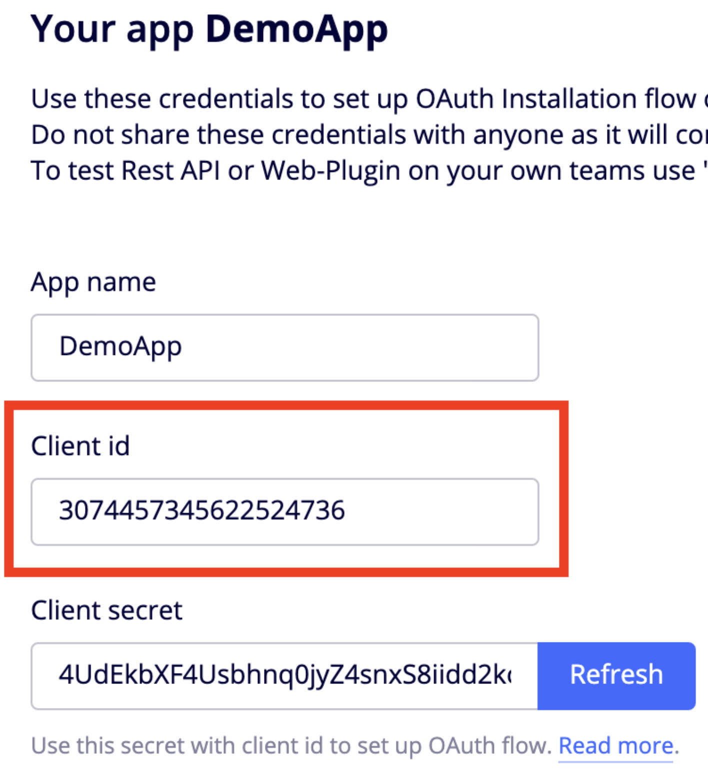
Figure 4. Example of an embedded Miro board with a specified viewpoint as the start view.

boardsPicker Component

How to configure boardsPicker:

1) Create a dev-team and an application.

Detailed instructions are here: https://developers.miro.com/docs/getting-started
You will need your application's **client_id** to open boardsPicker



Your app DemoApp

Use these credentials to set up OAuth Installation flow on your server.
Do not share these credentials with anyone as it will compromise your application.
To test Rest API or Web-Plugin on your own teams use 'Get OAuth Token' button below.

App name

DemoApp

Client id

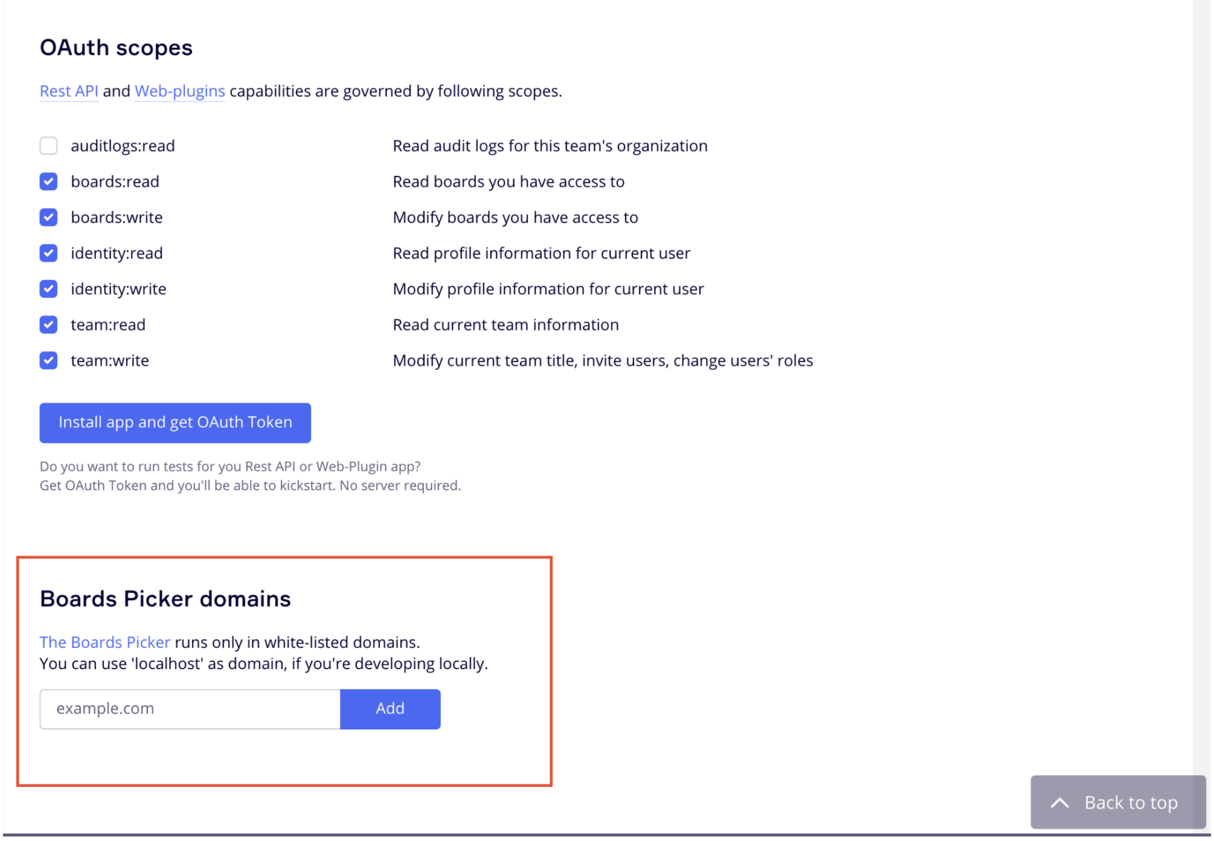
3074457345622524736

Client secret

4UdEkbXF4Usbhnq0jyZ4snxS8iidd2k

Use this secret with client id to set up OAuth flow. Read more.

2) In the app's settings, provide the domain names where your Boards Picker is hosted. This lets us stop other websites from trying to impersonate your app. If you're developing locally, you can use localhost as the domain name as well.



OAuth scopes

Rest API and Web-plugins capabilities are governed by following scopes.

☐ auditlogs:read	Read audit logs for this team's organization
☑ boards:read	Read boards you have access to
☑ boards:write	Modify boards you have access to
☑ identity:read	Read profile information for current user
☑ identity:write	Modify profile information for current user
☑ team:read	Read current team information
☑ team:write	Modify current team title, invite users, change users' roles

Do you want to run tests for you Rest API or Web-Plugin app?
Get OAuth Token and you'll be able to kickstart. No server required.

Boards Picker domains

The Boards Picker runs only in white-listed domains.
You can use 'localhost' as domain, if you're developing locally.

5) Add the following JavaScript snippet in your HTML.

JavaScript

```
<script type="text/javascript" src="https://miro.com/app/static/boardsPicker.1.0.js"></script>

<script>
  function onSomeButtonClick() {
    miroBoardsPicker.open({
      clientId: 'client_id_from_your_app',
      action: 'select',
      success: function (result) {
        console.log(result)
      }
    })
  }
</script>
```

Choose picker action [select ▾] [Open Boards Picker]

The source code of this demo is here.

> 🚧 Note that access-links generated in this example can't be used on your site because your site domain must be whitelisted.

Call the miroBoardsPicker.open() method to open the Boards Picker. Note that the Boards Picker opens in a pop-up window. Ensure to call this function from within a user-triggered event (click or tap). Otherwise, the browser blocks the pop-up.

The `open` method takes a single options parameter with the following fields:

Options

```
options = {
    // Required. ClientId from your app.
    clientId: 'string',

    // Required. Define mode of Boards Picker.
    action: "select" // or "access-link"

    // Required. Called when a user selects a board in the Picker.
    success: function(selectedBoard) {
        console.log(selectedBoard)
    },

    // Optional. Called when the user closes the pop-up.
    cancel: function() {
    },

    // Optional. Called when something went wrong.
    error: function() {
    },

    windowRef: windowRef, // Optional. Link to an already opened popup window. See the example below in ca
};
```

Choosing the action type

The boardsPicker component provides two types of action: `select` and `access-link`.

- `select` allows the user to choose a board and gives you a response with the board URL; you can use this URL, for example, to generate embed link;
- `access-link` allows the user to choose a board and choose an additional level of access for a unique link. With this action, you get a response with the board URL, access-link, and iFrame code that you can use for embedding;

Handling the response.

The success callback returns information about the selected board.

Result

```
//when the action is 'select'
selectedBoard = {
    id: "o9J_687XKf0="
    name: "Demo board"
    description: "Description for the demo board"
    viewLink: "https://miro.com/app/board/o9J_687XKf0=/"
}

//when the action id is 'acccess-link'
{
        accessLink: "https://miro.com/app/live-embed/o9J_k1ISaY4=?boardAccessToken=5IXCDJzAzw1bR85fJiYbH4W
        accessLinkPolicy: "EDIT"
        description: undefined
        embedHtml: "<iframe class=\"miro-embedded-board\" src=\"https://miro.com/app/live-embed/o9J_k1ISaY
        id: "o9J_k1ISaY4="
        name: "test"
        viewLink: "https://miro/app/board/o9J_k1ISaY4="
}
```

Enabling your integration for public use

By default, boardsPicker is only available for your Developer Team.

Submit this form when your integration is ready for public use. After submitting the form, you'll receive an email that will help you get going. In the meantime, we'll evaluate your request (usually within 24 - 48 hours) and we'll green-light your `clientId` for the relevant use cases.

Troubleshooting

If your website has a policy for blocking referrer information, Embeds via `access-link` does not load. To fix this, you can override the `referrerpolicy` attribute for the `iframe` tag on your site.

HTML

```html
<iframe src="https://miro.com/app/..." referrerpolicy="no-referrer-when-downgrade"></iframe>
```

Editable boards for not registered users

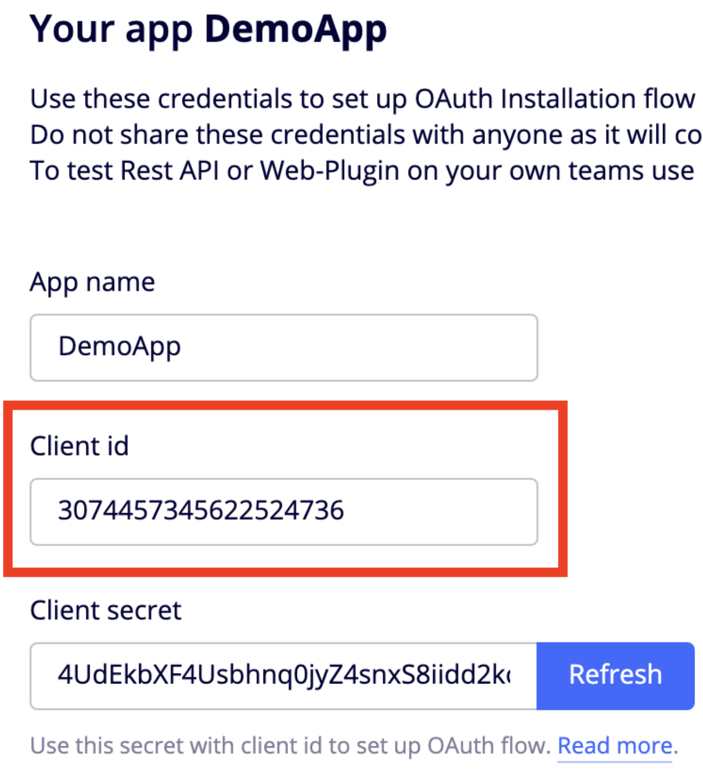
Miro Boards Picker allows the creation of boards for users who are not registered in Miro.

How to configure BoardsPicker:

1) Create a dev-team and application.

Detailed instructions here: https://developers.miro.com/docs/getting-started

2) Send us your clientId of the application. So we can enable BoardsPicker API.



Your app DemoApp

Use these credentials to set up OAuth Installation flow on your server.
Do not share these credentials with anyone as it will compromise your application.
To test Rest API or Web-Plugin on your own teams use 'Get OAuth Token' button below.

App name

DemoApp

Client id

3074457345622524736

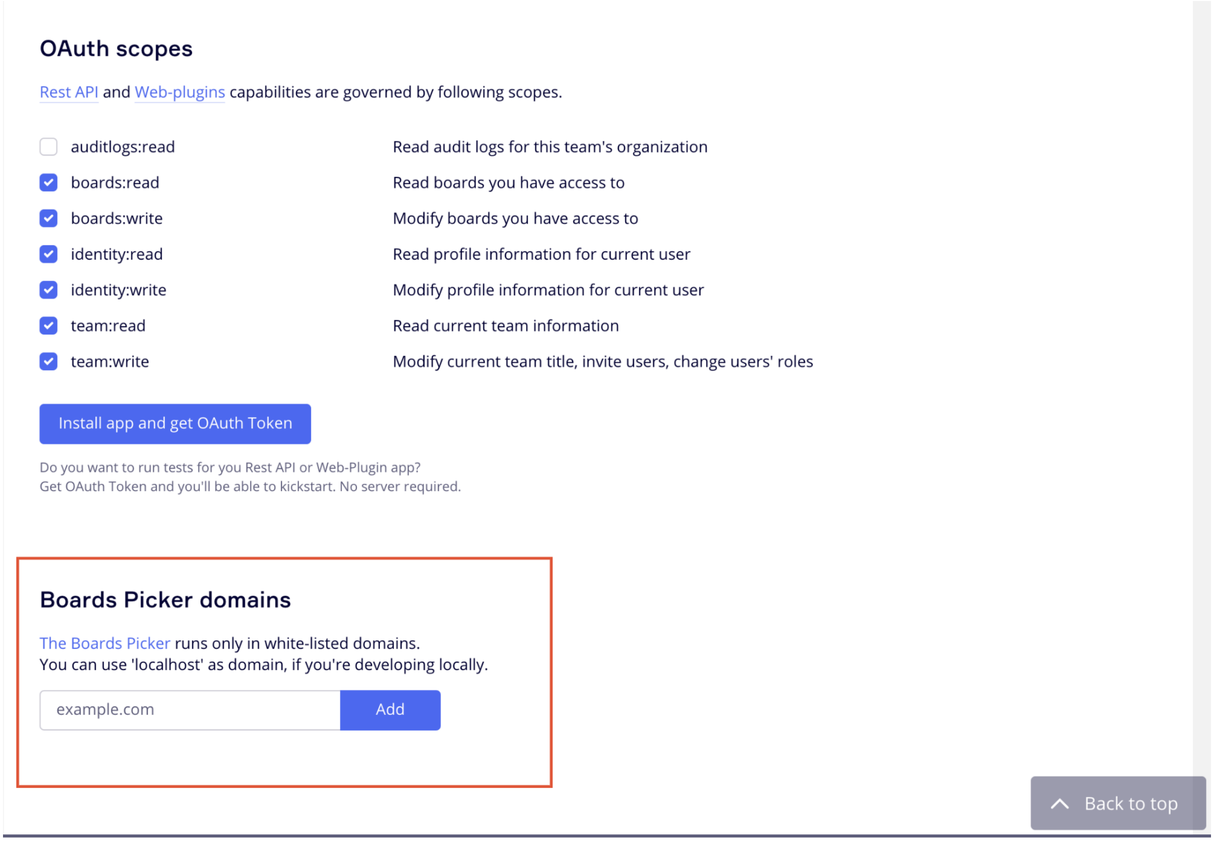
Client secret

4UdEkbXF4Usbhnq0jyZ4snxS8iidd2k

Use this secret with client id to set up OAuth flow. Read more.

3) [Use this guide](#) to configure basic BoardsPicker.

After this step, BoardsPicker should allow embedding boards for registered users.



OAuth scopes

Rest API and Web-plugins capabilities are governed by following scopes.

☐ auditlogs:read	Read audit logs for this team's organization
☑ boards:read	Read boards you have access to
☑ boards:write	Modify boards you have access to
☑ identity:read	Read profile information for current user
☑ identity:write	Modify profile information for current user
☑ team:read	Read current team information
☑ team:write	Modify current team title, invite users, change users' roles

Do you want to run tests for you Rest API or Web-Plugin app?
Get OAuth Token and you'll be able to kickstart. No server required.

Boards Picker domains

The Boards Picker runs only in white-listed domains.
You can use 'localhost' as domain, if you're developing locally.

example.com

4) Allow BoardsPicker to create boards for not registered users.

- Set `allowCreateAnonymousBoards=true`.
- Provide `JWT token` for BoardsPicker from your backend-side. This token used to create new boards on behalf of your integration.

JavaScript

```
function onClick() {
        miroBoardsPicker.open({
                clientId: '...', // 1) Put your 'clientId' here.
                action: 'access-link',
                allowCreateAnonymousBoards: true, //2) Enable this option
                getToken: () => getTokenFromServer(), // Provide token in async way
                success: data => {
                        console.log('on success', data)
                        document.querySelector('#container').innerHTML = data.embedHtml
                },
                error: e => {
                        console.log('on error', e)
                },
                cancel: () => {
                        console.log('on cancel')
                },

                windowRef: windowRef, // Optional. Link to an already opened popup window. See example bel
        })
}

// Type: () => Promise<string>
function getTokenFromServer() {
        //Get JWT token from your server. Read more about JWT https://jwt.io/
        return fetchPost('https://example.com/token-for-integration-with-miro')
}
```

5) Generate JWT on your backend.

- Get lib for your server language from https://jwt.io/
- Use **Client id** for **iss** field in payload data
- Use **Client secret** with **HMAC256** algorithm for Signing token
- We recommend setting a token expiration date of no more than 24 hours (**exp** field in payload data)

Example for Java using Auth0 library

```
JWT.create()
  .withIssuer("CLIENT_ID")
  .withExpiresAt(new Date(System.currentTimeMillis() + TimeUnit.MINUTES.toMillis(5)))
  .sign(Algorithm.HMAC256("CLIENT_SECRET"));
```

- Add endpoint like

```
POST https://example.com/token-for-integration-with-miro
```

This endpoint should return JWT

> ❗ Do not generate a token on the client-side and ensure that your endpoint is signed by your user authorization token. You can generate the JWT per timeout and you do not need to generate the JWT per user. JWTs are credentials that can grant access to resources. You must take all necessary precautions to keep JWTs safe.

Lazy loading boardsPicker.js example

JavaScript

```javascript
async function onClick() {

        //calc popup window size and position
        const pos = {left, top}
        const size = {width, height}

        const windowRef = window.open(
                '',
                '_blank',
                `width=${size.width},height=${size.height},left=${pos.left},top=${pos.top},menubar=0,toolb
        )

        await loadScript('https://miro.com/app/static/boardsPicker.1.0.js') // Lazy loading for boardsPick

        miroBoardsPicker.open({
                windowRef: windowRef, //Provide opened window reference
                clientId: '...',
                // ... rest params
        })
}
```

FAQs

How much does it cost to embed Miro into my website?

There is no price tag to embed Miro, but we review applications that embed Miro boards and boards picker. We expect these to be business, video conferencing, and education tools.

What are possible user flows?

You can implement one of the 2 flows:

- allow existent Miro users to embed their boards into your website,
- allow any user to create an empty Miro board without registering with Miro. The board will be deleted after 24 hours if the user does not move to one's own Miro team.

How can I get access?

Fill out this form to enter the waiting list of Miro embed partners. We will be in touch with you shortly.

How can I get help with Embed APIs and components?

For any technical questions about embed, reach out to us at our Developer Forum f

Getting Published in Miro Marketplace

Submit to the Miro Marketplace

We are on a mission to empower teams by providing them with apps and integrations that enrich collaboration through a self-serving ecosystem.

To ensure that, we opened the Miro Marketplace to developers who are willing to change the world of collaboration with us.

What apps are the best fit for the Miro Marketplace?

We welcome plugins and integrations that enrich collaboration for our key audience.
This means that your app must focus on user pain points within the following use cases:

- Facilitating or participating in workshops and remote meetings.
- Leading or following Agile rituals and workflows.
- Performing research and design methods.
- Developing software.
- Providing Miro users a better core user experience.

If you want to learn how our users solve their everyday collaboration tasks with Miro, follow our blog and Online Community, where customers and partners share stories of their projects, frameworks, and successes.

How do I get my app published?

Publishing an app means making it available for installation on the Miro Marketplace, which is available for millions of Miro users.

Here are the steps to get your app published in the Miro Marketplace:

- Check our design guidelines and ensure your app follows them. We want to deliver the best experience to users and expect developers to share our passion for great UX.
- Check our security guidelines and ensure your implementation follows them.
- **Submit your app through this form**.

The Miro Platform team will review your app and contact you about the next steps.

Getting your app design approved

We want to deliver the best experience to our users and expect developers to share our passion for great UX.

We have prepared detailed guidelines and a components library for you to use in your application. Ensure that you follow these requirements as we will be reviewing and approving your Marketplace publication based on these requirements.

We'd like you to build an app that:
• provides our users with excellent end-to-end experience,
• is as consistent with Miro UX as possible,
• is as neat as possible.

First, try to use one of the prebuilt templates. If there are no suitable templates, build your own using our components and design guidelines.

Security Guidelines

Introduction

We take great care of the security of our Miro product and our Marketplace in particular. Each new plugin and its subsequent updates go through a verification and approval process. We created this list of security requirements and recommendations to ensure the Miro Marketplace apps' high security and quality. This guideline focuses on the main pain-points: data privacy, performance impact, and vulnerabilities.

We separated the document into two parts: Requirements and Recommendations.

Requirements (main blockers that might affect the publication of your app)

Authentication and authorization

- Use OAuth 2.0 authorization. Don’t ask users for their credentials.
- Each request must be authenticated and authorized.
- You must authenticate and authorized access to Miro User Data that your App stores.

Session cookies

- `HttpOnly` & `Secure` flags must be set. You can detect the absence of these flags using a vulnerability scanner, such as Burp Suite.

Data storage

- Do not store Miro credentials.
- API keys must not be hardcoded in the source code.
- If you must store data, use high-load ready databases, S3.
- You must encrypt any confidential information that you store.
- Do not cache sensitive data. All HTTPS pages with sensitive data must use no-cache and no-store values in the Cache-Control header. You can check this by reviewing the request headers on pages with sensitive data. The Cache-Control header must look like this:
- HTTP/1.1:
- Cache-Control: no-cache, no-store
- Do not collect or store user data that is not necessary for the application to function.

Network configuration/Server configuration

- You are required to use TLS version 1.2 or higher to encrypt all application traffic. You can check your TLS version using Qualys SSL Labs in the Protocols section.
- There must be HTTPS-only connections on all App pages.
- HSTS must be enabled with a maximum age of at least one year. You can check your TLS version using Qualys SSL Labs in the Protocol Details section. The “Strict Transport Security (HSTS)” field must have the value of max-age greater than or equal to 31536000 and the "includeSubDomains" parameter must be enabled.

Domain Control

- TLS certificates on your domain must be valid. You can find the certificate with help of your browser or using Qualys SSL Labs. Your certificate must be signed by a Certificate Authority which is trusted by the major browsers.
- You must maintain control of domains and subdomains where your application is hosted.
- Your DNS configuration for subdomains must point to services that are in use.

Logging

- Ensure that secrets, tokens, and other sensitive information are not logged.

Performance

- Your App must not affect the whole Miro Service performance.

Vulnerabilities

- You must not use vulnerable libraries.
- There must not be medium, high, or critical vulnerabilities in your App. You can check this with OWASP Dependency-Check.

Recommendations (not blockers, but we suggest you follow these recommendations)

Authentication and authorization

- Grant scopes to your App in accordance with the principle of least privileged access.

Secure input and output handling

- Validate and sanitize all input to ensure that data is safe prior to use.
- Encode all output. The data must be processed as data and not as code.
- Avoid unrestricted file input.

Session cookies

- Session cookies must have high entropy so that they cannot be guessed or brute-forced.
- Ensure that session cookies are invalidated on logout.

Cryptography

- Use recommended algorithms, such as AES 256 in GCM mode. Do not create cryptographic algorithms.
- Initialize cryptographic keys using a secure random number generator.

Network configuration

- We recommend TLS version 1.2 using AES 256 encryption or higher with SHA-256 MAC.
- Ensure that HTTP methods, such as TRACE, are disabled if not being used.

Egress traffic

- Ensure that all egress traffic from your app is properly filtered and access to internal resources is restricted.
- Allowlist domains for egress traffic, if possible.
- Ensure that any domains or subdomains belonging to the App are owned and are not left dangling.
- For Connect apps, ensure that the baseUrl configured in the app’s descriptor file is valid and owned by the app.

Logging

- Log events and activities such as PRs being merged without approval and admin actions, such as creating or modifying existing users, new instances being created, admin logins, and alike.

Information leakage

- Do not expose OAuth tokens. OAuth client secrets must be handled carefully. Client secrets must not be distributed in emails, client-side Javascript, or error messages and must not be stored in public code repositories.
- Ensure that secrets are not leaked by the Referer header to third-parties: endpoints should implement a 302 found redirect. This is particularly important when app endpoints are handling authentication tokens.
- For any Connect apps, ensure that the shared secret is stored securely and not exposed on the client-side or server-side in error messages.

Design guidelines

Overview

To provide a better user experience for your users, it’s a good idea to use consistent UI and app components with the rest of the Miro product UI. While we are working on an official set of reusable components, we recommend using Mirotone CSS components—a base library that enables everyone to design and build their Miro apps. The goal of this design system is to enable you to quickly jump into a simple, consistent, and efficient user experience while creating your functional solutions.

Ready to design a great app experience? Explore Mirotone CSS components.

Examples and inspiration

Examples from Marketplace

Azure Cards

Azure Cards uses REST APIs and Web-plugins to sync cards on the board with Azure DevOps tasks.

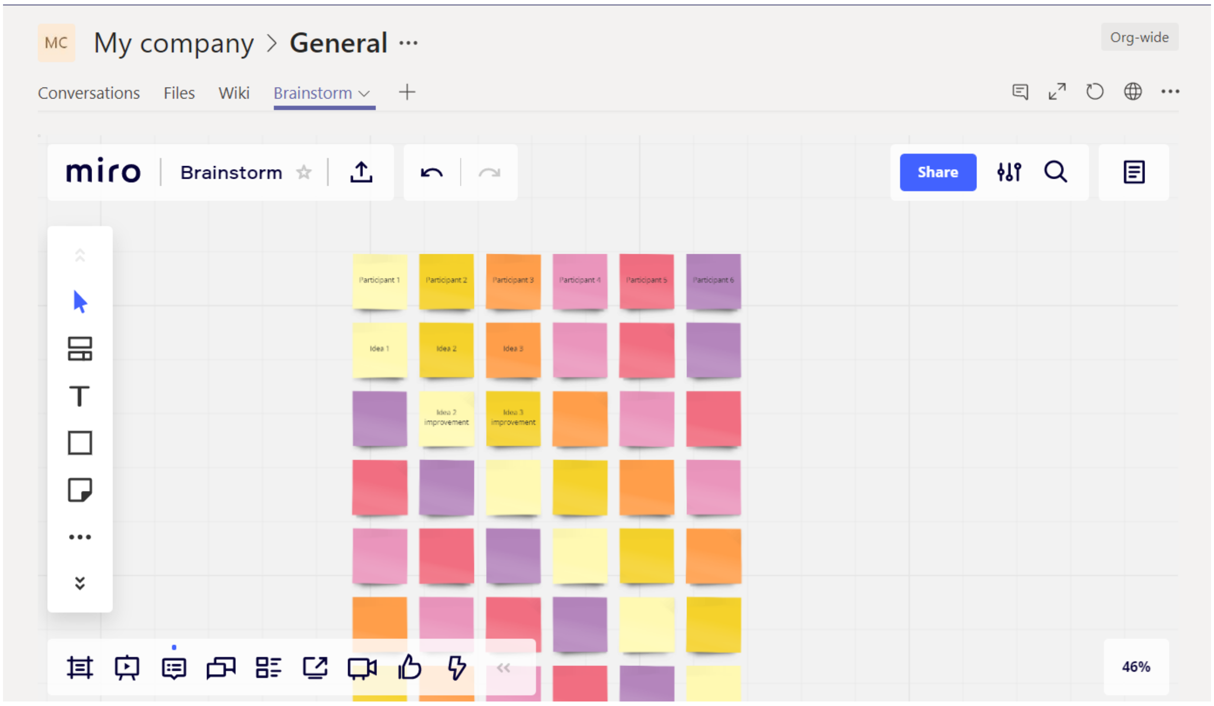
Learn more about this integration.

Microsoft Teams Bot

Microsoft Teams Bot:

- sends notifications about new comments.
- allows users to embed a board and collaborate on it right from the Microsoft Teams environment.

Learn more about this integration.



Examples with source code

You can find all examples in our Github.

Type of Task	Rest API	Web-Plugins
Import data to board	Import issues from GitHub	Template builder Turn stickies to shapes
Export data from board	Import issues from GitHub	Widget counter Looking glass Clear board
Manage users	Sync Miro user-list with Slack	

Ideas for inspiration

1. Board automation

Help Miro users work more quickly with board automation plugins. This should make manual routine work on the board simpler, faster, and more fun.

- Auto-format and highlight code so that developers can write and discuss code right in Miro (and even merge it right from there).
- It can be hard to make a board look tidy, with similar sizes and styles of widgets, the content aligned horizontally and vertically in a grid or line. It would be great to adjust a widget's position and make everything look nice with the click of a button.
- Hide and show layers of widgets, like in Photoshop.
- Sync widget values between each other, as different widgets often represent the same thing on one or several boards.

- Build charts via external libraries with data from boards or sheets.

2. Collaboration, agile rituals, and workshops

A lot of users come to a board to collaborate: whether it’s a workshop, daily standup, or brainstorming session. There are a lot of things you can do to make these moments more meaningful with additional tools and integrations.

- Breakout rooms for groups, each with its own frame.
- Bots for agile rituals and frameworks (retros, plannings, and so on) that guide users and tell them what to do.
- Plugin that would help during a planning poker exercise.
- Create a way to easily see who created a particular sticky right on it, automatically.
- Virtual dice, randomizer.

3. UX research and design

Looking for the right solution is key, but it’s also often the most time-consuming task. You can make this simpler by building plugins that allow users to analyze and clusterize data or build lo-fi prototypes faster.

Integrate icon-sets for faster wireframing and prototyping:

- Google Cloud Platform Iconset
- Shutterstock
- Giphy

Import and sync research data with external sources like:

- Google sheet
- Airtable
- Typeform

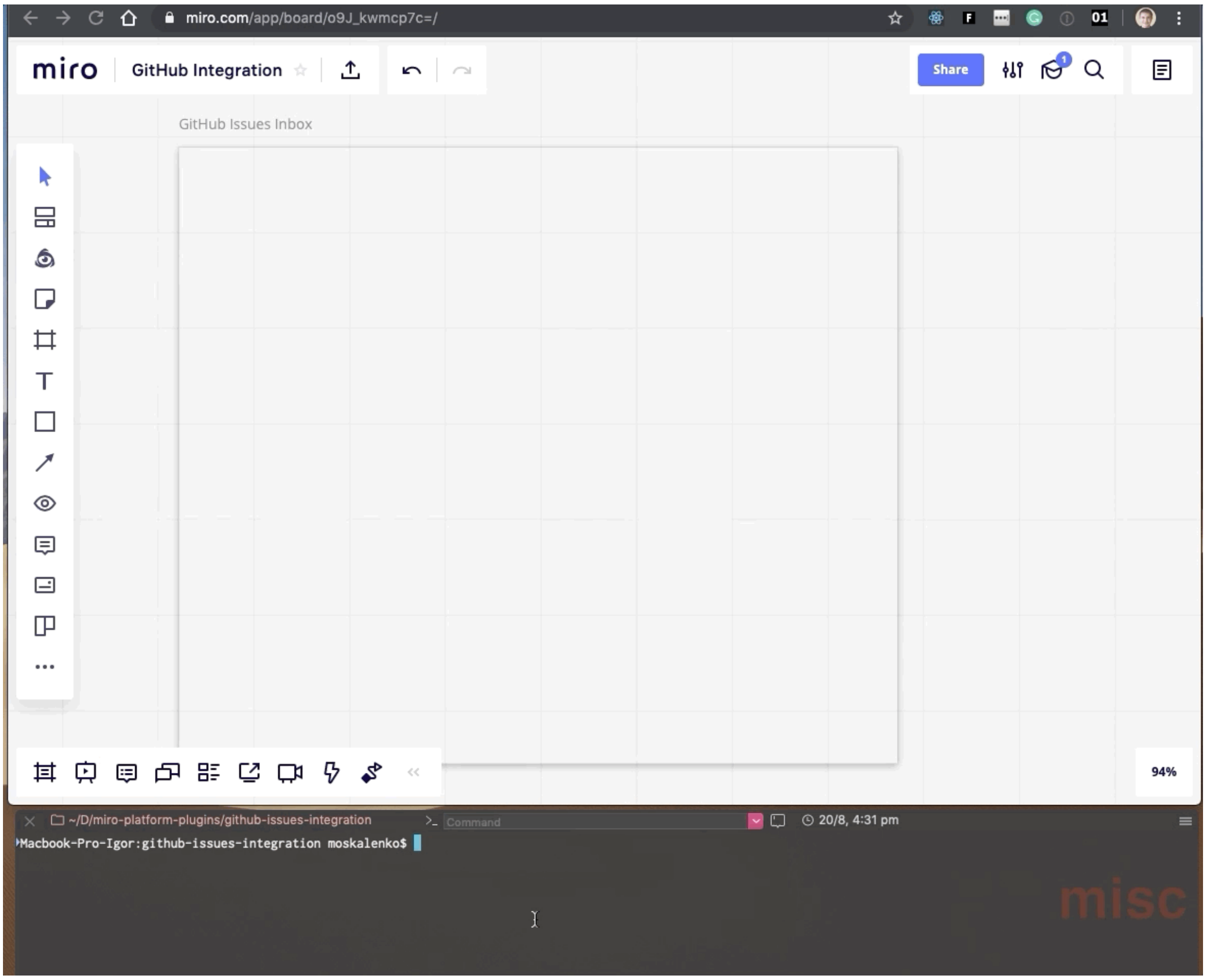
Simplify data processing by magic plugins:

- Cluster research data by keywords, color, and so on
- Build word cloud from selected content
- Smart import of sheet data into the board

Import data to board

Import issues from GitHub

This example shows how to import issues from GitHub to a board with custom fields.
It uses REST API for Board Widgets and Web-Plugins.


miro.com/app/board/o9J_kwmcp7c=/
miro
GitHub Integration
Share
GitHub Issues Inbox
94%
~/D/miro-platform-plugins/github-issues-integration
Command
20/8, 4:31 pm
misc

```
Macbook-Pro-Igor:github-issues-integration moskalenko$
```

Preparations

To start with this example, you must create a dev team and an app.
This guide shows you how to do it.

Used scopes

- board:write

Endpoints

- Create board widget

SourceCode

https://github.com/miroapp/app-examples/tree/v1/github-issue-importer

Web-plugins examples

We've provided three short examples of what and how you can build using web-plugins.
They cover almost all SDK features.

> 📘 These examples use the ES6 JavaScript version. Ensure you execute them in browsers that support this version.

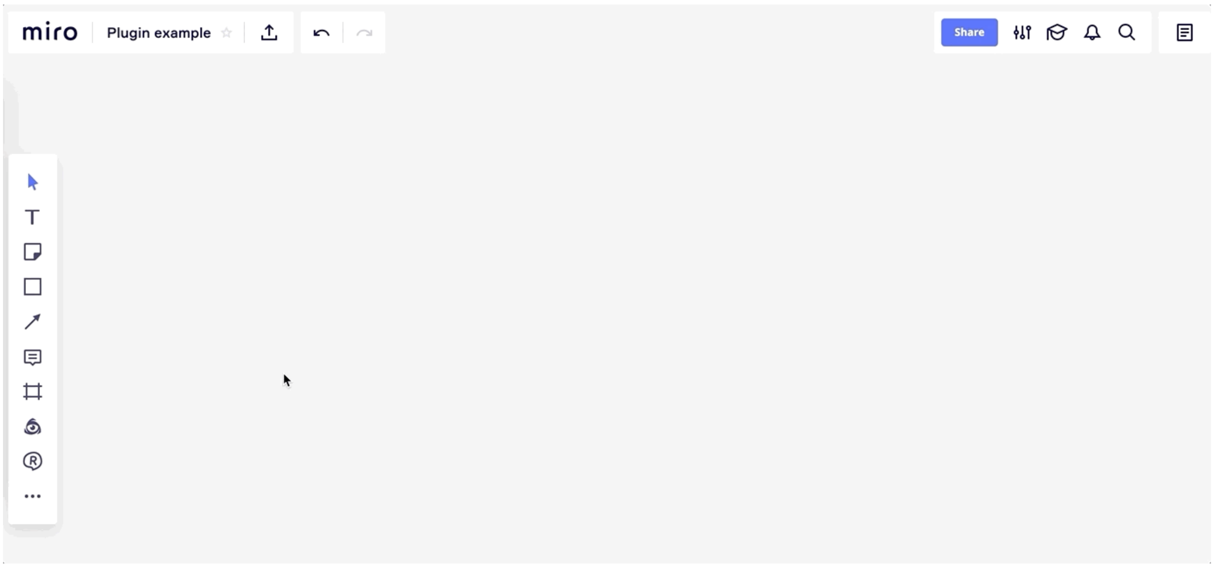
You can look at the example source code in app-examples repo.

- Template Builder
- Widget Counter

- Custom Icon Set with drag-and-drop
- Stickies to shapes
- Looking glass
- Clear board
- Spreadsheet sync

Template Builder

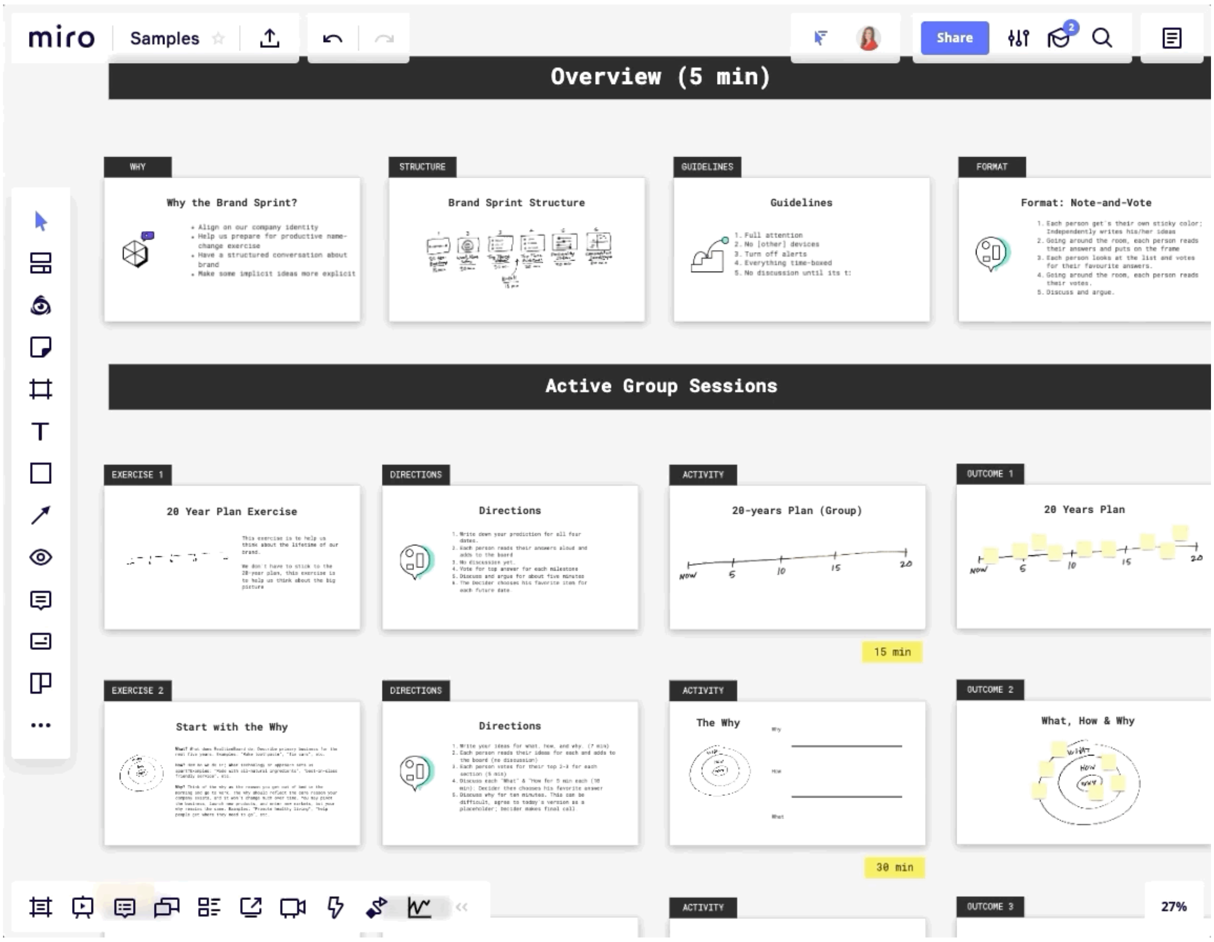
This is a sample Roadmap Template Builder you can build on Miro Web SDK. It shows how to create and position multiple widgets of different types on the board and create custom interfaces in the library.



Source code

Widget counter

You select several widgets, click the plugin button in the bottom bar, and the app shows the number of widgets grouped by type in the sidebar.



Why the Brand Sprint?

- Align on our company identity
- Help us prepare for productive name-change exercise
- Have a structured conversation about brand
- Make some implicit ideas more explicit

Guidelines

1. Full attention
2. No [other] devices
3. Turn off alerts
4. Everything time-boxed
5. No discussion until its t:

Format: Note-and-Vote

1. Each person get's their own sticky color; Independently writes his/her ideas
2. Going around the room, each person reads their answers and puts on the frame
3. Each person looks at the list and votes for their favourite answers.
4. Going around the room, each person reads their votes.
5. Discuss and argue.

20 Year Plan Exercise

This exercise is to help us think about the lifetime of our brand.

We don't have to stick to the 20-year plan, this exercise is to help us think about the big picture

Directions

1. Write down your prediction for all four dates.
2. Each person reads their answers aloud and adds to the board
3. No discussion yet.
4. Vote for top answer for each milestone
5. Discuss and argue for about five minutes
6. The Decider chooses his favorite item for each future date.

Start with the Why

Directions

1. Write your ideas for what, how, and why. (7 min)
2. Each person reads their ideas for each and adds to the board (no discussion)
3. Each person votes for their top 1-3 for each section (5 min)
4. Discuss each "What" & "How for 5 min each (10 min). Decider then chooses his favorite answer
5. Discuss why for ten minutes. This can be difficult, agree to today's version as a placeholder; Decider makes final call.

Used web-plugin features:

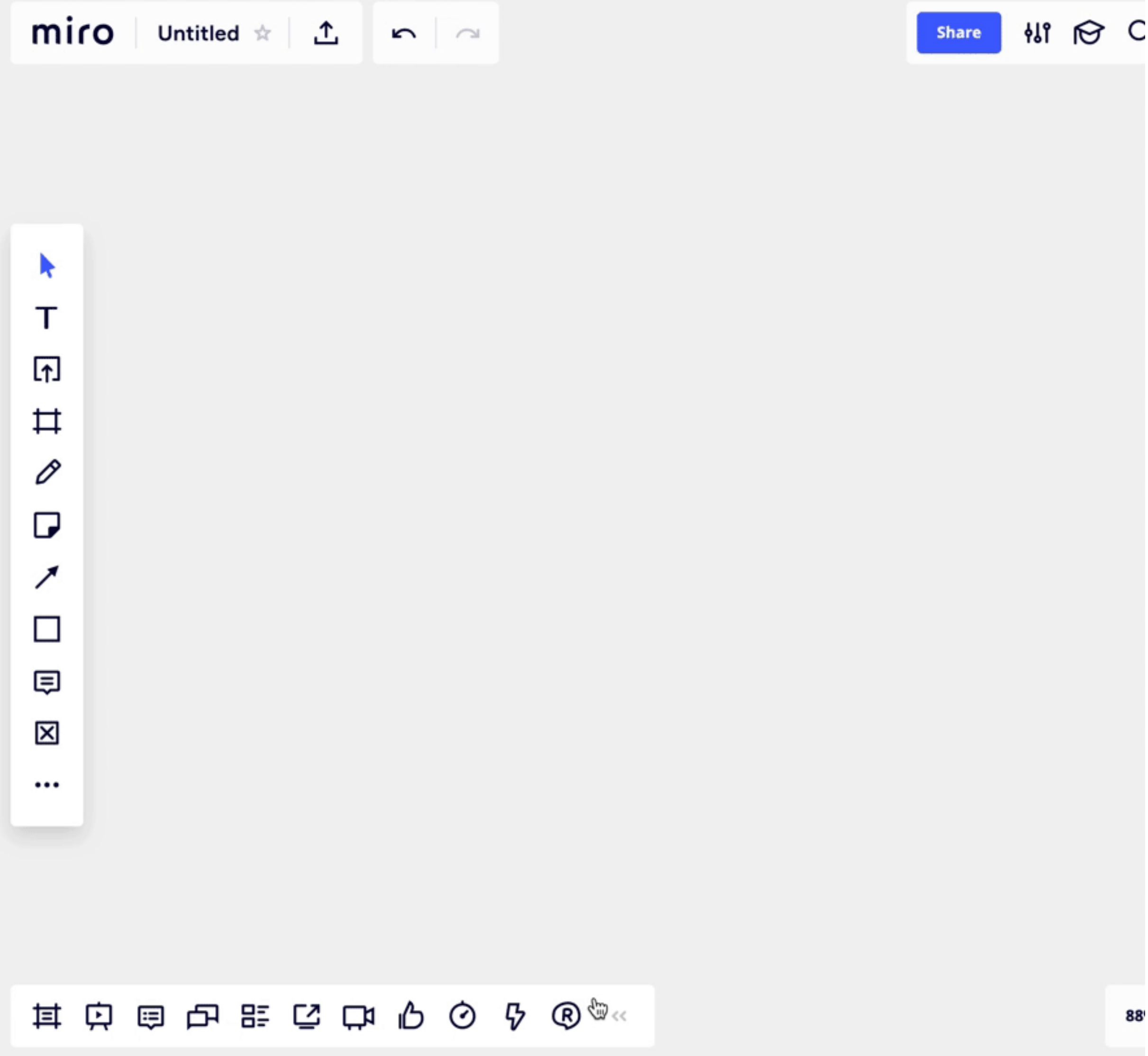
- Add button in bottom bar
- Get current selected widgets
- Subscribe to widgets selection updated event
- Open custom view in sidebar

SourceCode

Custom icon set with drag-and-drop

Used web-plugin features:

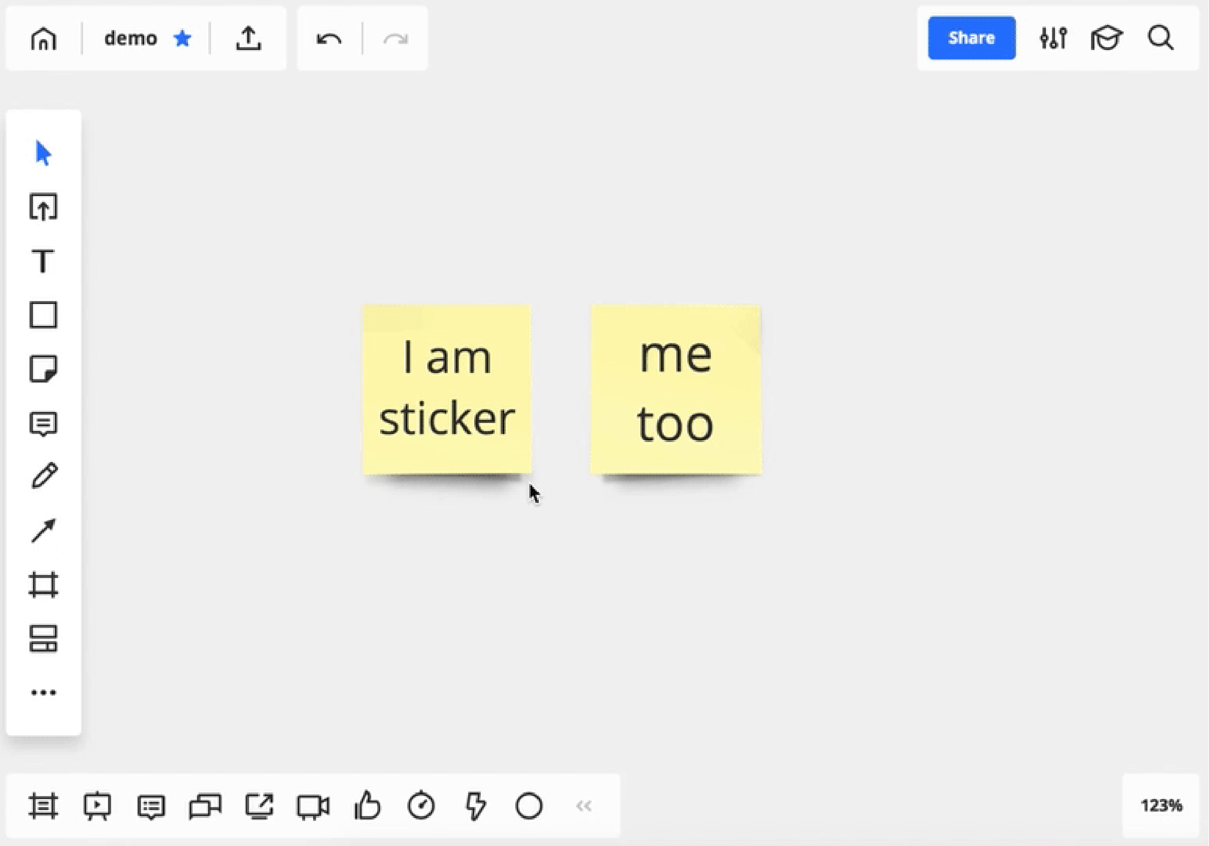
- Drag-and-drop images and shapes on the canvas
- Render custom view



SourceCode

Stickies to shapes

You select several stickies, click the plugin button in the bottom bar, and the app replaces stickies with shapes.



Used web-plugin features:

- Add button in bottom bar
- Get current selected widgets
- Remove widgets
- Create widgets

index.html

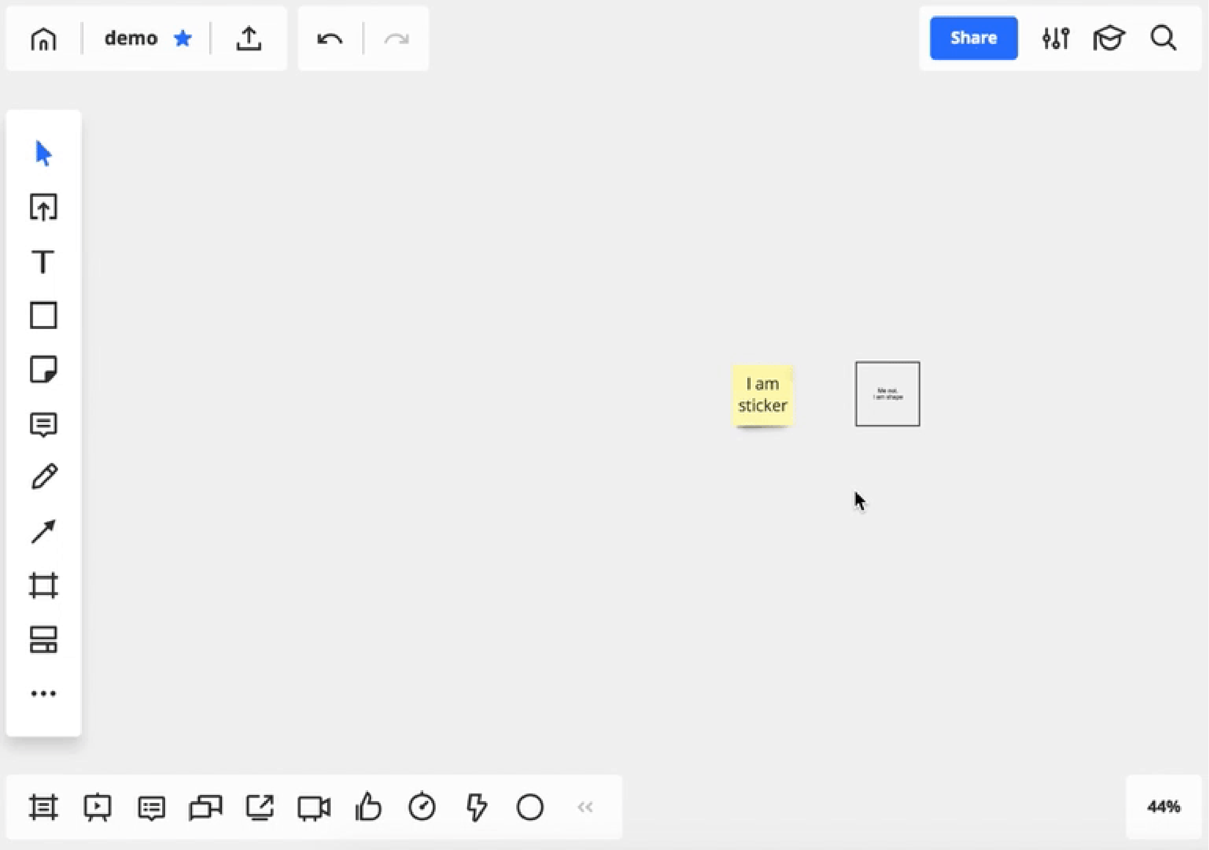
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://miro.com/app/static/sdk.1.1.js"></script>
    <script src="main.js"></script>
  </head>
</html>
```

main.js

```
miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      bottomBar: {
        title: 'Sticker to shapes',
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="none" fill-rule="evenodd" stroke="currentColor" stroke-widt
        positionPriority: 1,
        onClick: async () => {
          // Get selected widgets
          let selectedWidgets = await miro.board.selection.get()

          // Filter stickers from selected widgets
          let stickers = selectedWidgets.filter((widget) => widget.type === 'STICKER')

          // Delete selected stickers
          await miro.board.widgets.deleteById(stickers.map((sticker) => sticker.id))

          // Create shapes from selected stickers
          await miro.board.widgets.create(
            stickers.map((sticker) => ({
              type: 'shape',
              text: sticker.text,
              x: sticker.x,
              y: sticker.y,
              width: sticker.bounds.width,
              height: sticker.bounds.height,
            })),
          )

          // Show success message
          miro.showNotification('Stickers has been converted')
        },
      },
    },
  })
})
```

Looking glass

You can see the readable text of the widget in the sidebar without zooming in.



Used web-plugin features:

- Subscribe on events
- Open sidebar
- Show custom view in sidebar
- Add button in bottom bar
- Get current selected widgets

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://miro.com/app/static/sdk.1.1.js"></script>
    <script src="main.js"></script>
  </head>
</html>
```

main.js

```
let icon = '<circle cx="12" cy="12" r="9" fill="none" fill-rule="evenodd" stroke="currentColor" stroke-wid

miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      bottomBar: {
        title: 'Looking Glass',
        svgIcon: icon,
        positionPriority: 1,
        onClick: () => {
          miro.board.ui.openLeftSidebar('sidebar.html')
        },
      },
    },
  })
})
```

sidebar.html

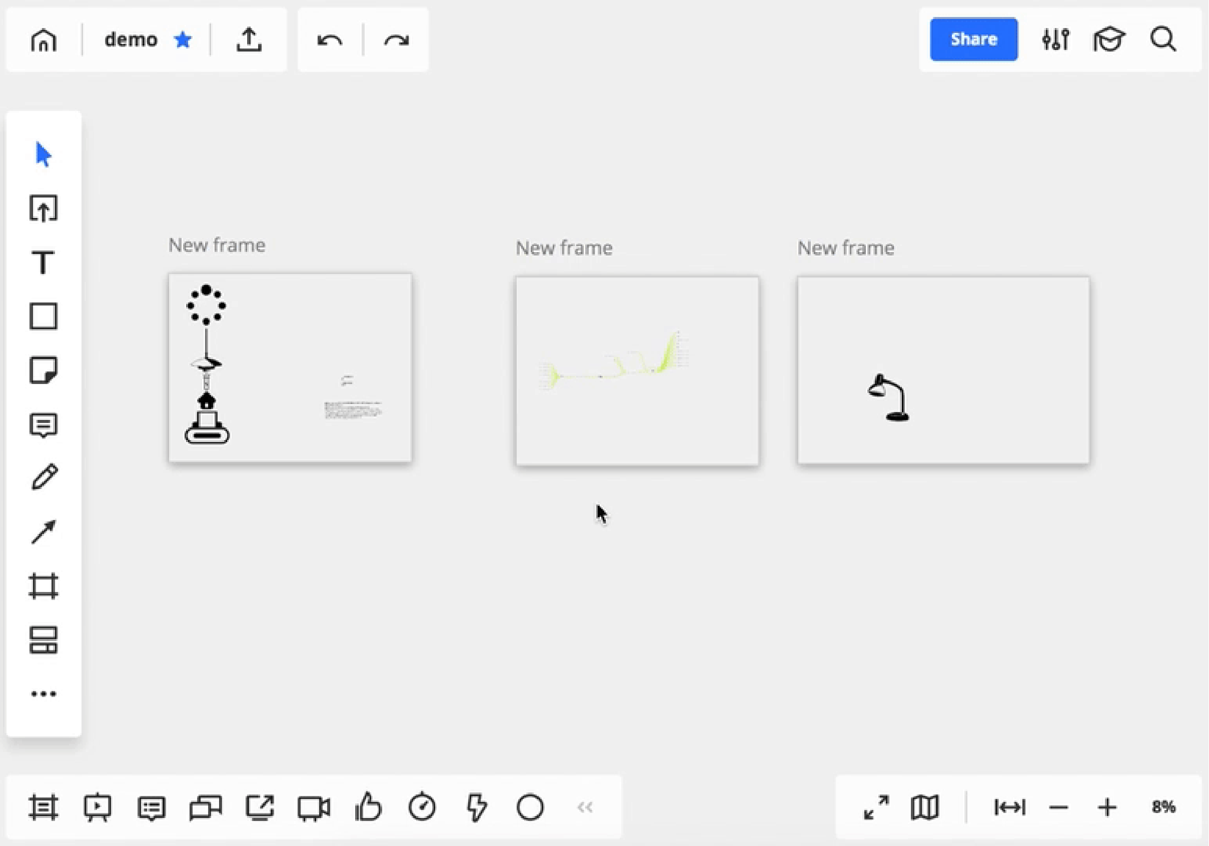
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <script src="https://miro.com/app/static/sdk.1.1.js"></script>

    <style>
      .rtb-sidebar-caption {
        font-size: 14px;
        font-weight: bold;
        color: rgba(0, 0, 0, 0.8);
        padding: 24px 0 0 24px;
      }

      textarea {
        display: block;
        height: calc(100% - 110px);
        margin: 20px 0 0 20px;
        background-color: white;
        border-radius: 4px;
        border: 0;
        width: calc(100% - 40px);
        text-align: left;
        font-size: 22px;
      }

      .tip {
        color: #cccccc;
        margin: 20px 0 0 26px;
      }
    </style>
  </head>

  <body>
    <div class="rtb-sidebar-caption">Looking Glass</div>
    <div class="tip" id="tip">Select widget to see its text</div>
    <textarea id="widget-text" readonly></textarea>

    <script src="sidebar.js"></script>
  </body>
</html>
```

sidebar.js

```
miro.onReady(() => {
  // Subscribe on user selected widgets
  miro.addListener(miro.enums.event.SELECTION_UPDATED, getWidget)
  getWidget()
})

// Get HTML elements for tip and text container
const tipElement = document.getElementById('tip')
const widgetTextElement = document.getElementById('widget-text')

async function getWidget() {
  // Get selected widgets
  let widgets = await miro.board.selection.get()

  // Get first widget from selected widgets
  let text = widgets[0].text

  // Check that the widget has text field
  if (typeof text === 'string') {
    // Hide tip and show text in sidebar
    tipElement.style.opacity = '0'
    widgetTextElement.value = text
  } else {
    // Show tip and clear text in sidebar
    tipElement.style.opacity = '1'
    widgetTextElement.value = ''
  }
}
```

Clear board

Delete all board content if user confirms the deletion.



Used web-plugin features:

- Open confirm modal
- Get all board objects
- Remove objects

index.html

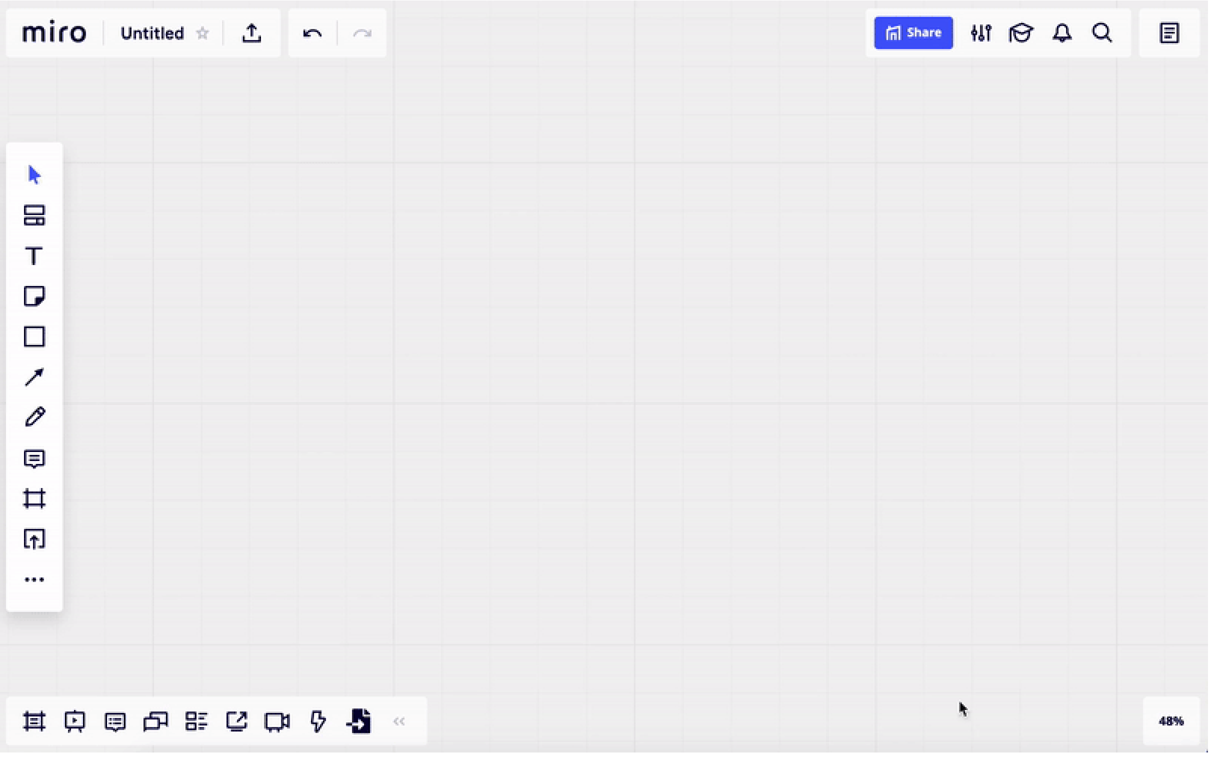
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://miro.com/app/static/sdk.1.1.js"></script>
    <script src="main.js"></script>
  </head>
</html>
```

main.js

```
const icon = '<circle cx="12" cy="12" r="9" fill="none" fill-rule="evenodd" stroke="currentColor" stroke-w

miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      bottomBar: {
        title: 'Board cleaner',
        svgIcon: icon,
        onClick: async () => {
          // Show modal and wait for user choice
          let needToClear = confirm('Do you want delete all content?')

          if (needToClear) {
            // Get all board objects
            let objects = await miro.board.widgets.get()

            // Delete all board objects
            await miro.board.widgets.deleteById(objects.map((object) => object.id))

            // Display success
            miro.showNotification('Content has been deleted')
          }
        },
      },
    },
  })
})
```

Spreadsheet sync

Sync data on the board with Google Sheets.



Source code

Add and remove users

Sync the list of team members from an external source

This example shows how to add and remove members from your team, based on data from an external source. For example, add new Miro users to a team when a new Miro user joins a specific Slack channel, and remove a user from your Miro team when the user leaves the Slack channel.

Preparation

To use this example, you need to create a dev team and an app.
This guide shows you how to do it.

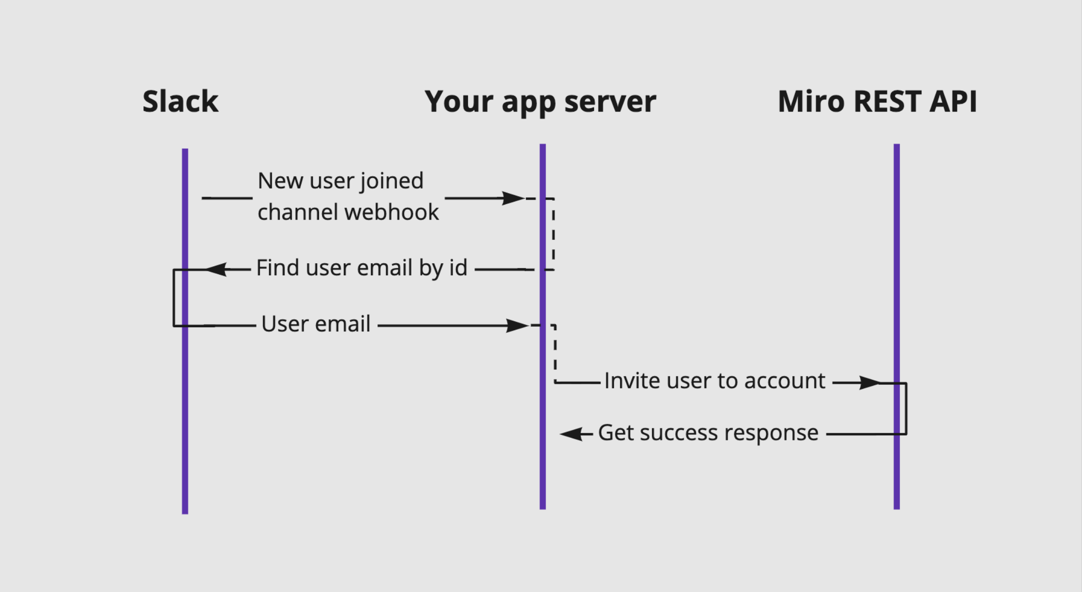
Used scopes

- team:read
- team:write

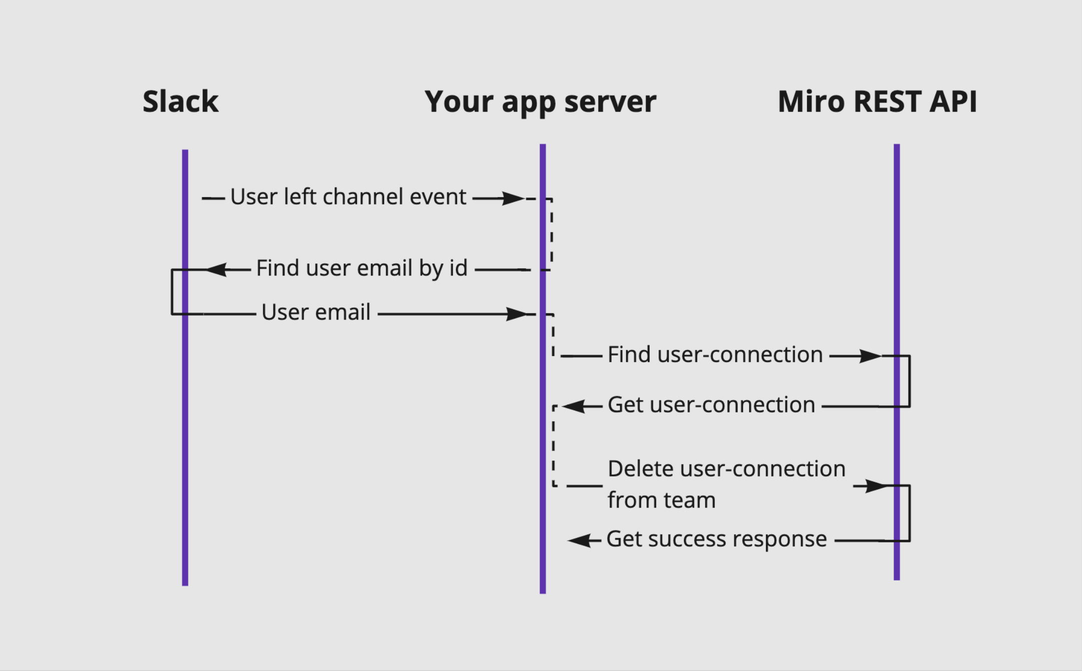
Endpoints

- Invite To Team
- Delete Board User Connection
- Get Board User Connection

Use case data flow



Case 1: Invite new slack users to Miro team



Case 2: Remove from Miro users that left Slack channel

SourceCode

https://github.com/miroapp/app-examples/tree/v1/automate-user-management

SDK Reference

Account and User

Account and User

Interface

- AuthorizationOptions
- IAccountCommands
- ICurrentUserCommands
- IUserInfo
- IAccountInfo
- IPictureInfo

interface-iuserinfo

Interface: IUserInfo

Hierarchy

- **IUserInfo**

Index

Properties

- email
- id
- name
- picture

Properties

email

• **email**: string

id

• **id**: string

name

• **name**: string

picture

• **picture**: IPictureInfo

interface-icurrentusercommands

Interface: ICurrentUserCommands

Commands related to the current user

Hierarchy

- **ICurrentUserCommands**

Index

Methods

Methods

getCurrentAccountPermissions

▸ **getCurrentAccountPermissions**(): Promise<AccountPermission[]>

Gets the permissions of the current user over the account (team).

Requires scope: `IDENTITY:READ`

Returns: Promise<AccountPermission[]>

a promise resolving into an array of AccountPermission(strings)

getCurrentBoardPermissions

▸ **getCurrentBoardPermissions**(): Promise<BoardPermission[]>

Gets the permissions of the current user over the current board.

Requires scope: `IDENTITY:READ`

Returns: Promise<BoardPermission[]>

a promise resolving into an array of BoardPermission(strings)

getId

‣ **getId**(): Promise<string>

Gets the user id

Returns: Promise<string>

a promise resolving in the current user id

getScopes

‣ **getScopes**(): Promise<string[]>

Gets the current scopes the user has authorized the plugin

Returns: Promise<string[]>

a promise resolving into an array of scopes (strings) the user has authorized the plugin.

isMemberOfCurrentAccount

‣ **isMemberOfCurrentAccount**(): Promise<boolean>

Returns if the current user is a member of the account (team) owner of the board

Requires scope: IDENTITY:READ

Returns: Promise<boolean>

a promise resolving into the membership status of the current user (boolean)

isSignedIn

‣ **isSignedIn**(): Promise<boolean>

Returns if the user is logged in.

Returns: Promise<boolean>

a promise resolved into the login status of the current user (true or false)

interface-iaccountcommands

Interface: IAccountCommands

Commands related to the account (team) on which the plugin
was installed.

Note: this is not the current user.

Hierarchy

- IAccountCommands

Index

Methods

- get

Methods

get

▸ **get**(): Promise<IAccountInfo>

Get information IAccountInfo about the account.

Returns: Promise<IAccountInfo>

a promise that resolves into the account information.

Requires scope: `TEAM:READ`

interface-authorizationoptions

Interface: AuthorizationOptions

Authorization options for the authorization flow.
See requestAuthorization

Hierarchy

- AuthorizationOptions

Index

Properties

- redirect_uri
- state

Properties

redirect_uri

• `Optional` **redirect_uri**: undefined | string

The url the user will be redirected after the authorization.
This url must be registered in your app "Redirect URLs" list.

state

- `Optional` **state**: undefined | string

oAuth state parameter. It will be send back to the defined `redirect_uri`.

interface-ipictureinfo

Interface: IPictureInfo

Hierarchy

- **IPictureInfo**

Index

Properties

- big
- image
- medium
- small

Properties

big

- **big**: string

image

- **image**: string

medium

- **medium**: string

small

- **small**: string

interface-iaccountinfo

Interface: IAccountInfo

Hierarchy

- **IAccountInfo**

Index

Properties

Properties

createdAt

• **createdAt**: string

---

currentUserPermissions

• **currentUserPermissions**: AccountPermission[]

---

id

• **id**: string

---

picture

• **picture**: IPictureInfo

---

title

• **title**: string

Board Manipulation

Board Manipulation

Interface

- IBoardCommands
- IBoardInfoCommands
- IBoardUtils
- IBoardViewportCommands
- IViewportOptions
- IBoardSelectionCommands
- IBoardCommentsCommands
- IBoardGroupsCommands

- IGroup
- IBoardInfo

interface-iboardcommentscommands

Interface: IBoardCommentsCommands

Hierarchy

- **IBoardCommentsCommands**

Index

Methods

- get

Methods

get

▸ **get**(): Promise<IComment[]>

Requires scope: BOARDS:READ

Returns: Promise<IComment[]>

interface-iboardcommands

Interface: IBoardCommands

Possible functions (commands) that you can execute
on the current board.

The specific commands and examples are defined on its own section.

Hierarchy

- **IBoardCommands**

Index

Properties

- info
- selection
- tags
- ui

- utils
- viewport
- widgets

Methods

- __disableLeftClickOnCanvas
- __enableLeftClickOnCanvas
- __getParamsFromURL

Properties

info

• **info**: IBoardInfoCommands

Command related to the board information

---

selection

• **selection**: IBoardSelectionCommands

Commands related to the selection of widgets in the board

---

tags

• **tags**: IBoardTagsCommands

Commands related to tags

---

ui

• **ui**: IBoardUICommands

Commands related to the user interface control

---

utils

• **utils**: IBoardUtils

Utilities to work with the board

---

viewport

• **viewport**: IBoardViewportCommands

Commands related to the board viewport

---

widgets

- **widgets**: IBoardWidgetsCommands

Commands related to widgets like create, update, delete and metadata.

Methods

__disableLeftClickOnCanvas

▸ **__disableLeftClickOnCanvas**(): void

Disable the usage of the mouse left button on the board.

`experimental`

Returns: void

__enableLeftClickOnCanvas

▸ **__enableLeftClickOnCanvas**(): void

Enables the usage of the mouse left button on the board.
It has no effect if the left button was not previously disabled.

`experimental`

Returns: void

__getParamsFromURL

▸ **__getParamsFromURL**(): Promise<any>

Returns the deserialized parameters contained in the `miro_sdk` address (location.href) query parameter.

`experimental`

Returns: Promise<any>

A promise resolving into the deserialized data in the `miro_sdk` query parameter.

interface-iboardutils

Interface: IBoardUtils

Utilities to work with a board.

Hierarchy

- **IBoardUtils**

Index

Methods

- unionWidgetBounds

Methods

unionWidgetBounds

▸ **unionWidgetBounds**(`widgets` : { bounds: IBounds }[]): IBounds

Calculates the union boundaries of several widgets.

Parameters:

Name	Type
`widgets`	{ bounds: IBounds }[]

Returns: IBounds

interface-iboardinfo

Interface: IBoardInfo

Hierarchy

- **IBoardInfo**

Index

Properties

- createdAt
- currentUserPermissions
- description
- id
- lastModifyingUser
- lastViewedByMeDate
- modifiedByMeDate
- owner
- picture
- title
- updatedAt

Properties

createdAt

• **createdAt**: string

currentUserPermissions

- **currentUserPermissions**: BoardPermission[]

description

- **description**: string

id

- **id**: string

lastModifyingUser

- **lastModifyingUser**: IUserInfo

lastViewedByMeDate

- **lastViewedByMeDate**: string

modifiedByMeDate

- **modifiedByMeDate**: string

owner

- `Optional` **owner**: IUserInfo

picture

- **picture**: IPictureInfo

title

- **title**: string

updatedAt

- **updatedAt**: string

interface-iviewportoptions

Interface: IViewportOptions

Options to set the board viewport

Hierarchy

- IViewportOptions

Index

Properties

- animationTimeInMS
- padding

Properties

animationTimeInMS

• `Optional` **animationTimeInMS**: undefined | number

Time in milliseconds for an animation effect.
Defaults to no animation.

padding

• `Optional` **padding**: IOffset

Padding between the target viewport and the final result
Defaults to 0

interface-iboardselectioncommands

Interface: IBoardSelectionCommands

Commands related to the selection and selected widgets in the board

Hierarchy

- **IBoardSelectionCommands**

Index

Methods

- clear
- enterSelectWidgetsMode
- get
- selectWidgets

Methods

clear

▸ **clear**(): Promise<void>

Unselect all widgets in the board

Returns: Promise<void>

A promised fulfilled if the action can be performed.

---

enterSelectWidgetsMode

▸ **enterSelectWidgetsMode**(): Promise<{ selectedWidgets: IWidget[] }>

Creates a dark mask and prompts the current user to select a widget in the board (widget selection mode).

The user can only select one widget at the time.

Note: This command is not available in container extension points.

Returns: Promise<{ selectedWidgets: IWidget[] }>

A promise resolving into the selected widgets.
If the user cancels the selection the array of selected widgets will be empty

---

get

▸ **get**(): Promise<IWidget[]>

Requires scope: BOARDS:READ

Returns: Promise<IWidget[]>

The currently selected widgets

---

selectWidgets

▸ **selectWidgets**(`widgetIds` : InputWidgets): Promise<IWidget[]>

Selects an specific widget.

Requires scope: BOARDS:READ

Parameters:

Name	Type
`widgetIds`	InputWidgets

Returns: Promise<IWidget[]>

The selected widgets

interface-iboardviewportcommands

Interface: IBoardViewportCommands

Commands to get information and manipulate the board viewport.

Hierarchy

- **IBoardViewportCommands**

Index

Methods

Methods

__mask

▸ **__mask**(`viewport` : IRect, `padding?` : IOffset): void

Adds a black mask over the canvas.

`experimental`

Parameters:

Name	Type
`viewport`	IRect
`padding?`	IOffset

Returns: void

__unmask

▸ **__unmask**(): void

Removes any mask set over the canvas.

`experimental`

Returns: void

get

▸ **get**(): Promise<IRect>

Returns information about the current viewport position

Returns: Promise<IRect>

getScale

▸ **getScale**(): Promise<number>

Returns the viewport scale (zoom level)

Returns: Promise<number>

set

▸ **set**(`viewport` : IRect, `options?` : IViewportOptions): Promise<IRect>

Allows to set the board viewport

Parameters:

Name	Type
`viewport`	IRect
`options?`	IViewportOptions

Returns: Promise<IRect>

zoomToObject

▸ **zoomToObject**(`widget` : InputWidget): Promise<void>

Zooms to a specific Widget on the board

Parameters:

Name	Type
`widget`	InputWidget

Returns: Promise<void>

interface-iboardinfocommands

Interface: IBoardInfoCommands

Commands to get information about the board

Hierarchy

- **IBoardInfoCommands**

Index

Methods

- get

Methods

get

▸ **get**(): Promise<IBoardInfo>

Returns: Promise<IBoardInfo>

a promise resolving into information (IBoardInfo) about the board

interface-iboardgroupscommands

Interface: IBoardGroupsCommands

Hierarchy

- **IBoardGroupsCommands**

Index

Methods

- get

Methods

get

▸ **get**(): Promise<IGroup[]>

Requires scope: BOARDS:READ

Returns: Promise<IGroup[]>

interface-igroup

Interface: IGroup

Hierarchy

- **IGroup**

Index

Properties

Properties

bounds

• **bounds**: IBounds

---

childrenIds

• **childrenIds**: string[]

---

id

• **id**: string

Extension Points

Extension Points

Interface

interface-iboarduicommands

Interface: IBoardUICommands

Commands to manipulate the user interface of the current board and make use of container extension points.

When a container extension point is opened (e.g. the sidebar) an iframe will be created and loaded based on the options you use to open it. There are different mechanism to communicate between these iframes, such events and states.

Read more about Extension Points

Hierarchy

- **IBoardUICommands**

Index

Methods

Methods

__clearToolbarModeLimit

▸ **__clearToolbarModeLimit**(): void

Removes any limitation set with __limitToolbarMode.

Note: this will not allow a non-authorized user to modify the board

Returns: void

__hideButtonsPanels

▸ **__hideButtonsPanels**(`panels` : "all" | UIPanel | UIPanel[]): void

Hides interface panels UIPanel

`experimental`

Parameters:

Name	Type
`panels`	"all" | UIPanel | UIPanel[]

Returns: void

__limitToolbarMode

▸ **__limitToolbarMode**(`mode` : "editor" | "commentor" | "viewer"): void

Sets the toolbar mode. Some modes show more options than other. Example: the `viewer` can only see the default tool.

Note: this will not limit the ability of the current user to interact with the board widgets.

Parameters:

Name	Type
`mode`	"editor" | "commentor" | "viewer"

Returns: void

__selectDefaultTool

▸ **__selectDefaultTool**(): void

Selects the board default tool

`experimental`

Returns: void

__showButtonsPanels

▸ **__showButtonsPanels**(`panels` : "all" | UIPanel | UIPanel[]): void

Shows interface panels UIPanel

Parameters:

Name	Type
`panels`	"all" | UIPanel | UIPanel[]

Returns: void

closeBottomPanel

▸ **closeBottomPanel**(`data?` : any): void

Closes the bottom panel.
Throws error if the bottom panel was not opened by the plugin

Parameters:

Name	Type	Description
`data?`	any	Ignored for now.

Returns: void

closeLeftSidebar

‣ **closeLeftSidebar**(`data?` : any): void

Closes the left sidebar.
Throws error if the sidebar was not opened by the plugin

Parameters:

Name	Type	Description
`data?`	any	Ignored for now.

Returns: void

closeLibrary

‣ **closeLibrary**(`data?` : any): void

Closes the library.
Throws error if the library was not opened by the plugin

Parameters:

Name	Type	Description
`data?`	any	Ignored for now.

Returns: void

closeModal

‣ **closeModal**(`data?` : any): void

Closes the modal.
Throws error if the modal was not opened by the plugin

Parameters:

Name	Type	Description
`data?`	any	Ignored for now.

Returns: void

initDraggableItemsContainer

▸ **initDraggableItemsContainer**(`container` : HTMLElement, `options` : DraggableItemsContainerOptions): void

Enables a container (HTMLElement) to have draggable elements (items).

Note: Not all items inside the container will be draggable.
See DraggableItemsContainerOptions to know how to define them.

Items dynamically added to the container will be draggable if matching the conditions defined in DraggableItemsContainerOptions

`todo` add link to a guide with a full working example

Parameters:

Name	Type	Description
`container`	HTMLElement	An HTMLElement containing the draggable items.
`options`	DraggableItemsContainerOptions	Define the conditions and behavior of the draggable items.

Returns: void

Example

JavaScript

```
//This example code should run in a container extension point like the left sidebar or the library.
miro.onReady(() => {
  // Contains the draggable elements
  const draggable = document.getElementById("draggables-container");
  miro.board.ui.initDraggableItemsContainer(draggable, {
    // matching elements will be draggable
    draggableItemSelector: ".drag-me",
    getDraggableItemPreview: function () {
      return {
        url: "https://dummyimage.com/600x400/000000/ffffff&text=Drag+Me",
      };
    },
    onDrop: function () {
      miro.showNotification("You dropped something");
    },
  });
});
```

---

openBottomPanel

▸ **openBottomPanel**(`iframeURL` : string, `options?` : undefined | { height?: undefined | number ; width?: undefined | number }): Promise<any>

Opens the bottom panel extension point and loads an iframe with the `iframeURL` .

You can communicate with this iframe by using the broadcastData method.

Parameters:

Name	Type	Description
`iframeURL`	string	the url that will be open in the iframe. If a relative url is used, it will be relative to the defined web plugin url.
`options?`	undefined | { height?: undefined | number ; width?: undefined | number }	Options for the bottomPanel. options.width: default 120px, min value: 80px; max: value 320px options.height: default 48px, min value: 48px; max: value 200px

Returns: Promise<any>

A promise that will be resolved once the bottomPanel is closed.

Example

JavaScript

```javascript
miro.onReady(async () => {
  miro.initialize({
    extensionPoints: {
      // create a button on the bottom bar
      bottomBar: {
        title: "Open a modal",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        onClick: () => {
          miro.board.ui.openBottomPanel("/sidebar.html", {
            width: 200,
            height: 200,
          });
        },
      },
    },
  });
});
```

openLeftSidebar

▸ **openLeftSidebar**(`iframeURL` : string): Promise<any>

Opens the left sidebar and loads an iframe with the `iframeURL` .

You can communicate with this iframe by using the broadcastData method.

Parameters:

Name	Type	Description
`iframeURL`	string	the url that will be open in the iframe. If a relative url is used, it will be relative to the defined web plugin url.

Returns: Promise<any>

A promise that will be resolved once the left sidebar is closed.

Example

JavaScript

```javascript
miro.onReady(async () => {
  miro.initialize({
    extensionPoints: {
      // create a button on the bottom bar
      bottomBar: {
        title: "Open a sidebar",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        onClick: () => {
          // open the sidebar when clicked
          // `/sidebar.html` will be relative to the plugin url
          miro.board.ui.openLeftSidebar("/sidebar.html");

          // send a message to the sidebar after 5 seconds
          setTimeout(() => {
            miro.broadcastData("Hello from the bottom bar");
          }, 5000);
        },
      },
    },
  });
});
```

openLibrary

▸ **openLibrary**(`iframeURL` : string, `options` : { title: string }): Promise<any>

Opens the library extension point and loads an iframe with the `iframeURL` .

You can communicate with this iframe by using the broadcastData method.

Parameters:

Name	Type	Description
`iframeURL`	string	the url that will be open in the iframe. If a relative url is used, it will be relative to the defined web plugin url.
`options`	{ title: string }	Options for the library. Current only `title` is available.

Returns: Promise<any>

A promise that will be resolved once the library is closed.

Example

JavaScript

```
miro.onReady(async () => {
  miro.initialize({
    extensionPoints: {
      // create a button on the bottom bar
      bottomBar: {
        title: "Open the library",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        onClick: () => {
          // `/library.html` will be relative to the plugin url
          miro.board.ui.openLibrary("/library.html", { title: "The library!" });
        },
      },
    },
  });
});
```

openModal

▸ **openModal**(`iframeURL` : string, `options?` : { height?: undefined | number ; width?: undefined | number } | { fullscreen: boolean }): Promise<any>

Opens the modal extension point and loads an iframe with the `iframeURL` .

You can communicate with this iframe by using the broadcastData method.

Parameters:

Name	Type	Description
`iframeURL`	string	the url that will be open in the iframe. If a relative url is used, it will be relative to the defined web plugin url.
`options?`	{ height?: undefined | number ; width?: undefined | number } | { fullscreen: boolean }	Options for the modal. You can define specific dimensions or make it fullscreen.

Returns: Promise<any>

A promise that will be resolved once the modal is closed.

Example

JavaScript

```
miro.onReady(async () => {
  miro.initialize({
    extensionPoints: {
      // create a button on the bottom bar
      bottomBar: {
        title: "Open a modal",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        onClick: () => {
          miro.board.ui.openModal("/modal.html", { width: 400, height: 400 }).then(() => {
            miro.showNotification("modal closed");
          });
        },
      },
    },
  });
});
```

resizeTo

▸ **resizeTo**(`value` : HTMLElement | string | { height?: undefined | number ; width?: undefined | number }): void

Resizes the current iframe inside the bottom-panel.

`todo` this method does not exist
resizeToInner does exist

The left sidebar and the modal are not yet supported.

Parameters:

Name	Type
`value`	HTMLElement | string | { height?: undefined | number ; width?: undefined | number }

Returns: void

interface-ipluginconfigextensionpoints

Interface: IPluginConfigExtensionPoints

Extensions points allow you to enhance the user interface with buttons.

You can define your own buttons using these extensions points, each extension points accepts a specific configuration and has its own limitations.

Some extensions points only allow you to add up to one button.

Read more about Extension Points

Hierarchy

- IPluginConfigExtensionPoints

Index

Properties

Properties

bottomBar

- `Optional` **bottomBar**: ButtonExtensionPoint<BottomBarButton>

The toolbar seen on the bottom left side of the board

You can directly define a BottomBarButton or a function returning a promise that resolves on one.

Example

JavaScript

```javascript
miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      bottomBar: {
        title: "The button title",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        onClick: () => {
          miro.showNotification("You clicked a bottom bar item!");
        },
      },
    },
  });
});
```

exportMenu

- `Optional` **exportMenu**: ButtonExtensionPoint<ExportMenuButton>

The toolbar seen when the user opens the export menu.

You can directly define a ExportMenuButton or a function returning a promise that resolves on one.

Example

JavaScript

```
miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      exportMenu: {
        title: "The button title",
        svgIcon:
          '<circle cx="12" cy="12" r="9" fill="red" fill-rule="evenodd" stroke="currentColor" stroke-width
        onClick: () => {
          miro.showNotification("You clicked a export menu item!");
        },
      },
    },
  });
});
```

getWidgetMenuItems

- `Optional` **getWidgetMenuItems**: undefined | (widgets: IWidget[], editMode: boolean) => Promise<IWidgetMenuItem | IWidgetMenuItem[]>

Deprecated
This method is deprecated.

`deprecated`

toolbar

- `Optional` **toolbar**: ButtonExtensionPoint<ToolbarButton>

The toolbar seen on the left side of the board

You can directly define a ToolbarButton or a function returning a promise that resolves on one.

Example

JavaScript

```javascript
miro.onReady(() => {
  miro.initialize({
    extensionPoints: {
      toolbar: {
        title: "The button title",
        librarySvgIcon:
          '<circle cx="12" cy="12" r="9" fill="blue" fill-rule="evenodd" stroke="currentColor" stroke-widt
        toolbarSvgIcon:
          '<circle cx="12" cy="12" r="9" fill="red" fill-rule="evenodd" stroke="currentColor" stroke-width
        onClick: () => {
          miro.showNotification("You clicked a toolbar item!");
        },
      },
    },
  });
});
```

interface-exportmenubutton

Interface: ExportMenuButton

Defines a button in the export menu extension point.
The export buttons appears after the user click the "export" button in a board.

See IPluginConfigExtensionPoints for examples.

Read more about Extension Points

Hierarchy

- **ExportMenuButton**

Index

Properties

- onClick
- svgIcon
- title

Properties

onClick

• **onClick**: () => void

A handler to be executed when the user clicks the button

svgIcon

• **svgIcon**: string

The icon that will be displayed.
Must be an SVGElement

---

title

• **title**: string

The title of the button., It will be displayed on mouse over.

See IPluginConfigExtensionPoints for examples.

interface-draggableitemscontaineroptions

Interface: DraggableItemsContainerOptions

Options to define the conditions and behavior of draggable items

Hierarchy

- **DraggableItemsContainerOptions**

Index

Properties

Properties

dragDirection

• `Optional` **dragDirection**: undefined | string

Allows to block the device (e.g. a tablet or smartphone) scrolling while the dragging takes place.
This option will have no effect on mouse-enabled devices like a desktop PC.

Possible values: `horizontal` and `vertical`
Defaults to `horizontal`

---

draggableItemSelector

• **draggableItemSelector**: string

A CSS selector. HTMLElement inside the container matching this selector will be draggable.

getDraggableItemPreview

• **getDraggableItemPreview**: (targetElement: HTMLElement) => { height?: undefined | number ; url: string ; width?: undefined | number }

A callback executed when the user starts dragging.

onCancel

• `Optional` **onCancel**: undefined | () => void

A callback executed when an item was dragged but not dropped in the board canvas.

onClick

• `Optional` **onClick**: undefined | (targetElement: HTMLElement) => void

A callback executed when a draggable item is clicked.
This callback will be executed regardless of the item being dragged or not.

onDrop

• **onDrop**: (canvasX: number, canvasY: number, targetElement: HTMLElement) => void

A callback executed when an item was dropped in the board canvas.
targetElement is the dragged and dropped item.

interface-toolbarbutton

Interface: ToolbarButton

Defines a button in the toolbar extension point.
The toolbar is displayed on the left side of the board.

By default your icon will be only displayed on the library.

See IPluginConfigExtensionPoints for examples.

Read more about Extension Points

Hierarchy

- **ToolbarButton**

Index

Properties

- librarySvgIcon
- onClick
- title
- toolbarSvgIcon

Properties

librarySvgIcon

• **librarySvgIcon**: string

The icon that will be displayed on the library (when clicking the dots `...` icon
in the toolbar).
Must be an SVGElement

By default your button will be displayed here.
The user must drag and drop you plugin icon from the library into the toolbar to display it directly in the toolbar.

---

onClick

• **onClick**: () => void

A handler to be executed when the user clicks the button

---

title

• **title**: string

The title of the button, It will be displayed on mouse over.

---

toolbarSvgIcon

• **toolbarSvgIcon**: string

The icon that will be displayed in the toolbar after the user drag it from the library.
Must be an SVG Element

interface-bottombarbutton

Interface: BottomBarButton

Defines a button in the bottom bar extension point.
The bottom bar is displayed on the bottom left side of the board.

See IPluginConfigExtensionPoints for examples.

Read more about Extension Points

Hierarchy

- **BottomBarButton**

Index

Properties

- onClick
- svgIcon
- title

Properties

onClick

• **onClick**: () => void

A handler to be executed when the user clicks the button

svgIcon

• **svgIcon**: string

The icon that will be displayed.
Must be an SVGElement

title

• **title**: string

The title of the button, It will be displayed on mouse over.

Styling

Styling

Enumeration

- ShapeType
- StickerType
- BorderStyle
- FontFamily
- TextAlign
- TextAlignVertical
- LineStyle
- LineType
- LineArrowheadStyle

Interface

- IEnums

enumeration-borderstyle

Enumeration: BorderStyle

Index

Enumeration members

- DASHED
- DOTTED
- NORMAL

Enumeration members

DASHED

• **DASHED**: = 1

DOTTED

• **DOTTED**: = 0

NORMAL

• **NORMAL**: = 2

enumeration-fontfamily

Enumeration: FontFamily

Index

Enumeration members

- ABRIL_FATFACE
- ARIAL
- BANGERS
- CAVEAT
- CURSIVE
- EB_GARAMOND
- FREDOKA_ONE

- GEORGIA
- GRADUATE
- GRAVITAS_ONE
- LEMON_TUESDAY
- NIXIE_ONE
- NOTO_SANS
- OPEN_SANS
- PERMANENT_MARKER
- PLEX_MONO
- PLEX_SANS
- PLEX_SERIF
- PT_SANS
- PT_SANS_NARROW
- PT_SERIF
- RAMMETTO_ONE
- ROBOTO
- ROBOTO_CONDENSED
- ROBOTO_MONO
- ROBOTO_SLAB
- TIMES_NEW_ROMAN
- TITAN_ONE

Enumeration members

ABRIL_FATFACE

- **ABRIL_FATFACE**: = 2

ARIAL

- **ARIAL**: = 0

BANGERS

- **BANGERS**:

CAVEAT

- **CAVEAT**:

CURSIVE

- **CURSIVE**: = 1

EB_GARAMOND

- EB_GARAMOND:

FREDOKA_ONE

- FREDOKA_ONE:

GEORGIA

- GEORGIA:

GRADUATE

- GRADUATE:

GRAVITAS_ONE

- GRAVITAS_ONE:

LEMON_TUESDAY

- LEMON_TUESDAY:

NIXIE_ONE

- NIXIE_ONE:

NOTO_SANS

- NOTO_SANS:

OPEN_SANS

- OPEN_SANS:

PERMANENT_MARKER

- PERMANENT_MARKER:

PLEX_MONO

- PLEX_MONO:

PLEX_SANS

- PLEX_SANS:

PLEX_SERIF

- **PLEX_SERIF**:

PT_SANS

- **PT_SANS**:

PT_SANS_NARROW

- **PT_SANS_NARROW**:

PT_SERIF

- **PT_SERIF**:

RAMMETTO_ONE

- **RAMMETTO_ONE**:

ROBOTO

- **ROBOTO**:

ROBOTO_CONDENSED

- **ROBOTO_CONDENSED**:

ROBOTO_MONO

- **ROBOTO_MONO**:

ROBOTO_SLAB

- **ROBOTO_SLAB**:

TIMES_NEW_ROMAN

- **TIMES_NEW_ROMAN**:

TITAN_ONE

- **TITAN_ONE**:

enumeration-linestyle

Enumeration: LineStyle

Index

Enumeration members

Enumeration members

DASHED

• **DASHED**: = 1

DOTTED

• **DOTTED**: = 4

NORMAL

• **NORMAL**: = 2

STRONG

• **STRONG**: = 3

enumeration-linetype

Enumeration: LineType

Index

Enumeration members

Enumeration members

ARROW

• **ARROW**: = 2

ARROW_SKETCH

• ARROW_SKETCH: = 9

LINE

• LINE: = 1

enumeration-textalignvertical

Enumeration: TextAlignVertical

Index

Enumeration members

- BOTTOM
- MIDDLE
- TOP

Enumeration members

BOTTOM

• BOTTOM: = "b"

MIDDLE

• MIDDLE: = "m"

TOP

• TOP: = "t"

interface-ienums

Interface: IEnums

Hierarchy

- IEnums

Index

Properties

- borderStyle
- fontFamily
- lineArrowheadStyle
- lineStyle
- lineType
- shapeType
- stickerType
- textAlign
- textAlignVertical

Properties

borderStyle

• **borderStyle**: *typeof* BorderStyle

fontFamily

• **fontFamily**: *typeof* FontFamily

lineArrowheadStyle

• **lineArrowheadStyle**: *typeof* LineArrowheadStyle

lineStyle

• **lineStyle**: *typeof* LineStyle

lineType

• **lineType**: *typeof* LineType

shapeType

• **shapeType**: *typeof* ShapeType

stickerType

• **stickerType**: *typeof* StickerType

textAlign

• **textAlign**: *typeof* TextAlign

textAlignVertical

• **textAlignVertical**: *typeof* TextAlignVertical

enumeration-stickertype

Enumeration: StickerType

Index

Enumeration members

Enumeration members

RECTANGLE

• RECTANGLE: = 1

SQUARE

• SQUARE: = 0

enumeration-shapetype

Enumeration: ShapeType

Index

Enumeration members

- PREDEFINED_PROCESS
- RECTANGLE
- RHOMBUS
- ROUNDER
- STAR
- TEXT_RECT
- TRAPEZE
- TRIANGLE

Enumeration members

ARROW_LEFT

• **ARROW_LEFT**: = 13

ARROW_LEFT_RIGHT

• **ARROW_LEFT_RIGHT**: = 21

ARROW_RIGHT

• **ARROW_RIGHT**: = 12

BARREL

• **BARREL**: = 26

BRACE_LEFT

• **BRACE_LEFT**: = 23

BRACE_RIGHT

• **BRACE_RIGHT**: = 24

BUBBLE

• **BUBBLE**: = 6

CIRCLE

• **CIRCLE**: = 4

CLOUD

• **CLOUD**: = 22

CROSS

- **CROSS**: = 25

HEXAGON

- **HEXAGON**: = 17

OCTAGON

- **OCTAGON**: = 18

PARALL

- **PARALL**: = 10

PENTAGON

- **PENTAGON**: = 16

PILL

- **PILL**: = 15

PREDEFINED_PROCESS

- **PREDEFINED_PROCESS**: = 20

RECTANGLE

- **RECTANGLE**: = 3

RHOMBUS

- **RHOMBUS**: = 8

ROUNDER

- **ROUNDER**: = 7

STAR

- **STAR**: = 11

TEXT_RECT

- **TEXT_RECT**: = 14

TRAPEZE

- **TRAPEZE**: = 19

TRIANGLE

- **TRIANGLE**: = 5

enumeration-linearrowheadstyle

Enumeration: LineArrowheadStyle

Index

Enumeration members

Enumeration members

ARC_ARROW

- **ARC_ARROW**: = 1

ARROW

- **ARROW**: = 6

CIRCLE

- **CIRCLE**: = 4

FILLED_ARROW

- **FILLED_ARROW**: = 8

FILLED_CIRCLE

- **FILLED_CIRCLE**: = 5

FILLED_RHOMBUS

- **FILLED_RHOMBUS**: = 3

NONE

- **NONE**: = 0

OPEN_ARROW

- **OPEN_ARROW**: = 7

RHOMBUS

- **RHOMBUS**: = 2

enumeration-textalign

Enumeration: TextAlign

Index

Enumeration members

- CENTER
- LEFT
- RIGHT

Enumeration members

CENTER

- **CENTER**: = "c"

LEFT

- **LEFT**: = "l"

RIGHT

- **RIGHT**: = "r"

Widgets Manipulation

Widgets Manipulation

Interface

- IBoardWidgetsCommands
- IBoardTagsCommands
- ITag
- IWidgetNamespaces
- IWidget
- ITextWidget
- IImageWidget
- IStickerWidget
- IShapeWidget
- ILineWidget
- IWebScreenshotWidget
- IFrameWidget
- ICurveWidget
- IEmbedWidget
- IPreviewWidget
- IEmojiWidget
- ICardWidget
- IDocumentWidget
- IMockupWidget
- IComment
- IWidgetShortData

interface-iwidgetnamespaces

Interface: IWidgetNamespaces

Hierarchy

- **IWidgetNamespaces**

  ↳ IWidget

Index

Properties

- capabilities
- clientVisible
- metadata

Properties

capabilities

- **capabilities**: WidgetCapabilities

clientVisible

- **clientVisible**: boolean

metadata

- **metadata**: WidgetMetadata

interface-iwebscreenshotwidget

Interface: IWebScreenshotWidget

Hierarchy

- IWidget

  ↳ **IWebScreenshotWidget**

Index

Properties

- bounds
- capabilities
- clientVisible
- createdUserId
- groupId
- id
- lastModifiedUserId
- metadata
- scale
- type
- url
- x
- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

scale

- **scale**: number

type

- **type**: "WEBSCREEN"

Overrides IWidget.type

url

- `Readonly` **url**: string

x

- **x**: number

y

- **y**: number

interface-icomment

Interface: IComment

Hierarchy

- IWidget

  ↳ **IComment**

Index

Properties

- bounds
- capabilities
- clientVisible
- color
- createdUserId
- groupId
- id
- lastModifiedUserId
- metadata
- resolved
- type

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

color

- **color**: number

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

resolved

- **resolved**: boolean

type

• **type**: "COMMENT"

Overrides IWidget.type

interface-iboardwidgetscommands

Interface: IBoardWidgetsCommands

Commands related to widgets such creation, modification and metadata.

Hierarchy

- **IBoardWidgetsCommands**

Index

Methods

- __blinkWidget
- __getIntersectedObjects
- areAllWidgetsLoaded
- bringForward
- create
- deleteById
- get
- sendBackward
- transformDelta
- update

Methods

__blinkWidget

▸ **__blinkWidget**(`widgets` : InputWidgets): Promise<void>

Makes a widget perform a "blink" animation in the board.

`experimental`

Parameters:

Name	Type
`widgets`	InputWidgets

Returns: Promise<void>

A promise resolving if the widget was found.

__getIntersectedObjects

▸ **__getIntersectedObjects**(`pointOrRect` : IPoint | IRect): Promise<IWidget[]>

Finds all widgets intersected in a specific area.

Requires scope: BOARDS:READ

`experimental`

Parameters:

Name	Type
`pointOrRect`	IPoint | IRect

Returns: Promise<IWidget[]>

A Promise resolving with the widgets found in the requested area.

areAllWidgetsLoaded

▸ **areAllWidgetsLoaded**(): Promise<boolean>

Checks if all widgets on the board had been loaded.

Returns: Promise<boolean>

A promise resolving after all the widgets in the board had been loaded.

bringForward

▸ **bringForward**(`widgetId` : InputWidgets): Promise<void>

Brings a widget forward in the board (above other widgets)

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type
`widgetId`	InputWidgets

Returns: Promise<void>

A promise resolving after the widget was brought forward.

create

▸ **create**<T>(`widgets` : OneOrMany<WidgetToBeCreated>): Promise<T[]>

Creates a widget in the board.
A `type` is required.

See IWidget for more information about the different type of widgets and their properties.

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for the current user

Example

```javascript
miro.board.widgets.create({type: "sticker", text: "Hello world"})
```

Type parameters:

Name	Type
`T`	IWidget

Parameters:

Name	Type
`widgets`	OneOrMany<WidgetToBeCreated>

Returns: Promise<T[]>

A promise resolving into an array containing the newly created widgets.

deleteById

▸ **deleteById**(`widgetIds` : InputWidgets): Promise<void>

Deletes a single widget from the board

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type	Description
`widgetIds`	InputWidgets	Widget to be deleted

Returns: Promise<void>

A promise resolving after the widget was deleted.

get

▸ **get**<T>(`filterBy?` : Record<string, unknown>): Promise<T[]>

Get all the board widgets matching a filter `filterBy` .
filterBy uses https://lodash.com/docs/4.17.11#filter

See IWidget for more information about the different type of widgets and their properties.

Requires scope: BOARDS:READ

Example

JavaScript

```
const stickers = miro.board.widgets.get({type: "sticker"})
```

Type parameters:

Name	Type
`T`	IWidget

Parameters:

Name	Type
`filterBy?`	Record<string, unknown>

Returns: Promise<T[]>

A promise resolving into an array of widgets matching the filter

sendBackward

▸ **sendBackward**(`widgetId` : InputWidgets): Promise<void>

Send a widget to the back (below other widgets).

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type
`widgetId`	InputWidgets

Returns: Promise<void>

A promise resolving after the widget was sent forward.

transformDelta

▸ **transformDelta**(`widgetIds` : InputWidgets, `deltaX?` : undefined | number, `deltaY?` : undefined | number, `deltaRotation?` : undefined | number): Promise<IWidget[]>

Modifies the widgets position and rotation based on the passed parameters.

Parameters:

Name	Type	Description
`widgetIds`	InputWidgets	-
`deltaX?`	undefined | number	Translation on the X axis.
`deltaY?`	undefined | number	Translation on the Y axis
`deltaRotation?`	undefined | number	Rotation of the widget Requires scope: BOARDS:WRITE Requires BoardPermission.EDIT_CONTENT for current user

Returns: Promise<IWidget[]>

A promise resolving into an array with the modified widgets.

update

▸ **update**<T>(`widgets` : OneOrMany<{ [index:string]: any; id: string }>): Promise<T[]>

Modify one or many widgets in the board.
An `id` is mandatory.

See IWidget for more information about the different type of widgets and their properties.

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Type parameters:

Name	Type
`T`	IWidget

Parameters:

Name	Type
`widgets`	OneOrMany<{ [index:string]: any; id: string }>

Returns: Promise<T[]>

A promise resolving into an array with the modified widgets.

interface-iboardtagscommands

Interface: IBoardTagsCommands

Commands to interact and manipulate tags

Hierarchy

- **IBoardTagsCommands**

Index

Methods

- create
- delete
- get
- update

Methods

create

▸ **create**(`tags` : OneOrMany<CreateTagRequest>): Promise<ITag[]>

Creates one or many tags as defined in the CreateTagRequest.

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type	Description
`tags`	OneOrMany<CreateTagRequest>	Details of the tag being created. It may include what widgets will have the tag. A `title` is always required.

Returns: Promise<ITag[]>

A promise resolving into the created tags.

---

delete

▸ **delete**(`tags` : InputTags): Promise<void>

Deletes one or many tags.

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type	Description
`tags`	InputTags	The tags to be deleted.

Returns: Promise<void>

A promise resolving when the tags are deleted.

get

▸ **get**(`filterBy?` : Record<string, unknown>): Promise<ITag[]>

Gets all tags in the board, optionally filtered as specific in `filterBy`

Requires scope: BOARDS:READ

Parameters:

Name	Type	Description
`filterBy?`	Record<string, unknown>	Conditions to filter tags.

Returns: Promise<ITag[]>

A promise resolving into the found tags.

update

▸ **update**(`tags` : OneOrMany<UpdateTagRequest>): Promise<ITag[]>

Update one or many tags as defined in the UpdateTagRequest

Requires scope: BOARDS:WRITE
Requires BoardPermission.EDIT_CONTENT for current user

Parameters:

Name	Type	Description
`tags`	OneOrMany<UpdateTagRequest>	The tags to be updated. An `id` is always required.

Returns: Promise<ITag[]>

A promise resolving into the updated tags.

interface-iframewidget

Interface: IFrameWidget

Hierarchy

- IWidget

  ↳ **IFrameWidget**

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

childrenIds

- `Readonly` **childrenIds**: string[]

---

clientVisible

- **clientVisible**: boolean

Inherited from [IWidgetNamespaces](#).[clientVisible](#)

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

frameIndex

- `Readonly` **frameIndex**: number

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

height

- **height**: number

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

style

- **style**: { backgroundColor: BackgroundColorStyle }

Type declaration:

Name	Type
`backgroundColor`	BackgroundColorStyle

title

- **title**: string

type

- **type**: "FRAME"

Overrides IWidget.type

width

- **width**: number

x

- **x**: number

y

- **y**: number

interface-itag

Interface: ITag

Defines a tag and its properties

Hierarchy

- **ITag**

Index

Properties

- color
- id
- title
- widgetIds

Properties

color

- **color**: string | number

The color of the tag

id

• **id**: string

title

• **title**: string

widgetIds

• **widgetIds**: string[]

A list of ids with the assigned tag.

interface-icardwidget

Interface: ICardWidget

Hierarchy

- IWidget

  ↳ **ICardWidget**

Index

Properties

- title
- type
- x
- y

Properties

assignee

- `Optional` **assignee**: undefined | { userId: string }

---

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

card

- **card**: { customFields?: { fontColor?: undefined | string ; iconUrl?: undefined | string ; mainColor?: undefined | string ; roundedIcon?: undefined | false | true ; value?: undefined | string }[] ; logo?: undefined | { iconUrl: string } }

Type declaration:

Name	Type
`customFields?`	{ fontColor?: undefined | string ; iconUrl?: undefined | string ; mainColor?: undefined | string ; roundedIcon?: undefined | false | true ; value?: undefined | string }[]
`logo?`	undefined | { iconUrl: string }

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

date

- `Optional` **date**: undefined | string

Date format: `YYYY-mm-dd`

description

- **description**: string

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

rotation

- **rotation**: number

scale

- **scale**: number

style

- **style**: { backgroundColor: BackgroundColorStyle }

Type declaration:

Name	Type
`backgroundColor`	BackgroundColorStyle

tags

- `Readonly` **tags**: ITag[]

title

- **title**: string

type

- **type**: "CARD"

Overrides IWidget.type

x

- **x**: number

y

- **y**: number

interface-iwidgetshortdata

Interface: IWidgetShortData

Hierarchy

- **IWidgetShortData**

Index

Properties

Properties

id

- **id**: string

metadata

- `Optional` **metadata**: any

type

- `Optional` **type**: undefined | string

interface-iemojiwidget

Interface: IEmojiWidget

Hierarchy

- IWidget

  ↳ **IEmojiWidget**

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

code

- `Readonly` **code**: string

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.[id](id)

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

rotation

- **rotation**: number

scale

- **scale**: number

type

• **type**: "EMOJI"

Overrides IWidget.type

x

• **x**: number

y

• **y**: number

interface-iembedwidget

Interface: IEmbedWidget

Hierarchy

- IWidget

  ↳ **IEmbedWidget**

Index

Properties

- bounds
- capabilities
- clientVisible
- createdUserId
- groupId
- html
- id
- lastModifiedUserId
- metadata
- scale
- type
- x
- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

html

- **html**: string

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

scale

• **scale**: number

type

• **type**: "EMBED"

Overrides IWidget.type

x

• **x**: number

y

• **y**: number

interface-icurvewidget

Interface: ICurveWidget

Hierarchy

- IWidget

  ↳ **ICurveWidget**

Index

Properties

- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

---

id

- `Readonly` **id**: string

Inherited from IWidget.id

---

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

---

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

points

- **points**: IPoint[]

rotation

- **rotation**: number

scale

- **scale**: number

style

- **style**: { lineColor: LineColorStyle ; lineWidth: LineThicknessStyle }

Type declaration:

Name	Type
`lineColor`	LineColorStyle
`lineWidth`	LineThicknessStyle

type

- **type**: "CURVE"

Overrides IWidget.type

x

- **x**: number

y

- **y**: number

interface-itextwidget

Interface: ITextWidget

Hierarchy

- IWidget

↳ ITextWidget

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

---

id

- `Readonly` **id**: string

Inherited from IWidget.id

---

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

---

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

---

rotation

- **rotation**: number

---

scale

- **scale**: number

---

style

- **style**: { backgroundColor: BackgroundColorStyle ; backgroundOpacity: BackgroundOpacityStyle ; bold: BoldStyle ; borderColor: BorderColorStyle ; borderOpacity: BorderOpacityStyle ; borderStyle: BorderStyle ; borderWidth: BorderWidthStyle ; fontFamily: FontFamily ; highlighting: HighlightingStyle ; italic: ItalicStyle ; padding: PaddingStyle ; strike: StrikeStyle ; textAlign: TextAlign ; textColor: TextColorStyle ; underline: UnderlineStyle }

Type declaration:

Name	Type
`backgroundColor`	BackgroundColorStyle
`backgroundOpacity`	BackgroundOpacityStyle
`bold`	BoldStyle
`borderColor`	BorderColorStyle
`borderOpacity`	BorderOpacityStyle
`borderStyle`	BorderStyle
`borderWidth`	BorderWidthStyle
`fontFamily`	FontFamily
`highlighting`	HighlightingStyle
`italic`	ItalicStyle
`padding`	PaddingStyle
`strike`	StrikeStyle
`textAlign`	TextAlign
`textColor`	TextColorStyle
`underline`	UnderlineStyle

text

- **text**: string

type

- **type**: "TEXT"

Overrides IWidget.type

width

- **width**: number

x

- **x**: number

y

- **y**: number

interface-idocumentwidget

Interface: IDocumentWidget

Hierarchy

- IWidget

  ↳ **IDocumentWidget**

Index

Properties

Properties

bounds

• `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

• **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

• **clientVisible**: boolean

Inherited from [IWidgetNamespaces.clientVisible]

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

rotation

- **rotation**: number

scale

- **scale**: number

title

- **title**: string

type

- **type**: "DOCUMENT"

Overrides IWidget.type

x

- **x**: number

y

- **y**: number

interface-ipreviewwidget

Interface: IPreviewWidget

Hierarchy

- IWidget

  ↳ **IPreviewWidget**

Index

Properties

- bounds
- capabilities
- clientVisible
- createdUserId
- groupId
- id
- lastModifiedUserId
- metadata
- scale
- type
- url
- x
- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

scale

- **scale**: number

type

- **type**: "PREVIEW"

Overrides IWidget.type

url

- `Readonly` **url**: string

---

x

- **x**: number

---

y

- **y**: number

interface-iimagewidget

Interface: IImageWidget

Hierarchy

- IWidget

  ↳ **IImageWidget**

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

rotation

- **rotation**: number

scale

- **scale**: number

title

- **title**: string

type

- **type**: "IMAGE"

Overrides IWidget.type

url

- **url**: string

x

- **x**: number

y

- **y**: number

interface-imockupwidget

Interface: IMockupWidget

Hierarchy

- IWidget

  ↳ **IMockupWidget**

Index

Properties

- bounds
- capabilities
- clientVisible
- createdUserId
- groupId
- id
- lastModifiedUserId

- metadata
- mockupType
- rotation
- type
- x
- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

---

id

- `Readonly` **id**: string

Inherited from IWidget.id

---

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from [IWidget](#).[lastModifiedUserId](#)

metadata

• **metadata**: [WidgetMetadata](#)

Inherited from [IWidgetNamespaces](#).[metadata](#)

mockupType

• `Readonly` **mockupType**: string

rotation

• **rotation**: number

type

• **type**: "MOCKUP"

Overrides [IWidget](#).[type](#)

x

• **x**: number

y

• **y**: number

interface-istickerwidget

Interface: IStickerWidget

Hierarchy

- [IWidget](#)

  ↳ **IStickerWidget**

Index

Properties

- [bounds](#)
- [capabilities](#)
- [clientVisible](#)

- createdUserId
- groupId
- id
- lastModifiedUserId
- metadata
- plainText
- scale
- style
- tags
- text
- type
- x
- y

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

---

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

plainText

- **plainText**: string

This text does not include HTML characters.

scale

- **scale**: number

style

- **style**: { fontFamily: FontFamily ; fontSize: FontSizeStyle ; stickerBackgroundColor: BackgroundColorStyle ; stickerType: StickerType ; textAlign: TextAlign ; textAlignVertical: TextAlignVertical }

Type declaration:

Name	Type
`fontFamily`	FontFamily
`fontSize`	FontSizeStyle
`stickerBackgroundColor`	BackgroundColorStyle
`stickerType`	StickerType
`textAlign`	TextAlign
`textAlignVertical`	TextAlignVertical

tags

- `Readonly` **tags**: ITag[]

text

- **text**: string

This text will include HTML characters if present.

---

type

- **type**: "STICKER"

Overrides IWidget.type

---

x

- **x**: number

---

y

- **y**: number

interface-iwidget

Interface: IWidget

Defines a basic widget and its properties

Hierarchy

- IWidgetNamespaces

  ↳ **IWidget**

  ↳↳ ITextWidget

  ↳↳ IImageWidget

  ↳↳ IStickerWidget

  ↳↳ IShapeWidget

  ↳↳ ILineWidget

  ↳↳ IWebScreenshotWidget

  ↳↳ IFrameWidget

  ↳↳ ICurveWidget

  ↳↳ IEmbedWidget

  ↳↳ IPreviewWidget

↳↳ IEmojiWidget

↳↳ ICardWidget

↳↳ IDocumentWidget

↳↳ IMockupWidget

↳↳ IComment

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

---

id

- `Readonly` **id**: string

---

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

---

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

---

type

- `Readonly` **type**: string

interface-ilinewidget

Interface: ILineWidget

Defines a widget of type `LINE` .

Currently line widgets can only be created between two widgets.
// The `startWidgetId` and `endWidgetId` fields are required to create a line widget.

Hierarchy

- IWidget

  ↳ **ILineWidget**

Index

Properties

- metadata
- startPosition
- startWidgetId
- style
- type

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

---

captions

- `Readonly` **captions**: { text: string }[]

---

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

---

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

---

endPosition

- `Readonly` **endPosition**: IPoint

---

endWidgetId

- **endWidgetId**: string | undefined

Mandatory field

---

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

startPosition

- `Readonly` **startPosition**: IPoint

startWidgetId

- **startWidgetId**: string | undefined

Mandatory field

style

- **style**: { lineColor: LineColorStyle ; lineEndStyle: LineArrowheadStyle ; lineStartStyle: LineArrowheadStyle ; lineStyle: LineStyle ; lineThickness: LineThicknessStyle ; lineType: LineType }

Type declaration:

Name	Type
`lineColor`	LineColorStyle
`lineEndStyle`	LineArrowheadStyle
`lineStartStyle`	LineArrowheadStyle
`lineStyle`	LineStyle
`lineThickness`	LineThicknessStyle
`lineType`	LineType

type

- **type**: "LINE"

Overrides IWidget.type

interface-ishapewidget

Interface: IShapeWidget

Hierarchy

- IWidget

  ↳ **IShapeWidget**

Index

Properties

Properties

bounds

- `Readonly` **bounds**: IBounds

Inherited from IWidget.bounds

---

capabilities

- **capabilities**: WidgetCapabilities

Inherited from IWidgetNamespaces.capabilities

clientVisible

- **clientVisible**: boolean

Inherited from IWidgetNamespaces.clientVisible

createdUserId

- `Readonly` **createdUserId**: string

Inherited from IWidget.createdUserId

groupId

- `Optional` `Readonly` **groupId**: undefined | string

Inherited from IWidget.groupId

height

- **height**: number

id

- `Readonly` **id**: string

Inherited from IWidget.id

lastModifiedUserId

- `Readonly` **lastModifiedUserId**: string

Inherited from IWidget.lastModifiedUserId

metadata

- **metadata**: WidgetMetadata

Inherited from IWidgetNamespaces.metadata

plainText

- **plainText**: string

This text does not include HTML characters.

rotation

- **rotation**: number

style

- **style**: { backgroundColor: BackgroundColorStyle ; backgroundOpacity: BackgroundOpacityStyle ; bold: BoldStyle ; borderColor: BorderColorStyle ; borderOpacity: BorderOpacityStyle ; borderStyle: BorderStyle ; borderWidth: BorderWidthStyle ; fontFamily: FontFamily ; fontSize: FontSizeStyle ; highlighting: HighlightingStyle ; italic: ItalicStyle ; shapeType: ShapeType ; strike: StrikeStyle ; textAlign: TextAlign ; textAlignVertical: TextAlignVertical ; textColor: TextColorStyle ; underline: UnderlineStyle }

Type declaration:

Name	Type
`backgroundColor`	BackgroundColorStyle
`backgroundOpacity`	BackgroundOpacityStyle
`bold`	BoldStyle
`borderColor`	BorderColorStyle
`borderOpacity`	BorderOpacityStyle
`borderStyle`	BorderStyle
`borderWidth`	BorderWidthStyle
`fontFamily`	FontFamily
`fontSize`	FontSizeStyle
`highlighting`	HighlightingStyle
`italic`	ItalicStyle
`shapeType`	ShapeType
`strike`	StrikeStyle
`textAlign`	TextAlign
`textAlignVertical`	TextAlignVertical
`textColor`	TextColorStyle
`underline`	UnderlineStyle

text

- **text**: string

This text will include HTML characters if present.

type

• **type**: "SHAPE"

Overrides IWidget.type

width

• **width**: number

x

• **x**: number

y

• **y**: number

interface-event

Interface: Event

An Event generated by the main application.

You can subscribe to events using addListener

Hierarchy

- **Event**

Index

Properties

- data
- type

Properties

data

• **data**: any

Payload with data related to the event

type

• **type**: string | EventType

The type of event as in [[EventIBoardCommentsCommandspe]]

interface-ibounds

Interface: IBounds

Hierarchy

- **IBounds**

Index

Properties

Properties

bottom

• **bottom**: number

height

• **height**: number

left

• **left**: number

right

• **right**: number

top

• **top**: number

width

• **width**: number

x

• **x**: number

y

• **y**: number

interface-ioffset

Interface: IOffset

Hierarchy

- IOffset

Index

Properties

- bottom
- left
- right
- top

Properties

bottom

• **bottom**: number

left

• **left**: number

right

• **right**: number

top

• **top**: number

interface-ipluginconfig

Interface: IPluginConfig

Defines the configuration to initialize the web plugin
See [`IPluginConfigExtensionPoints`] for further information.

Hierarchy

- **IPluginConfig**

Index

Properties

- extensionPoints

Properties

extensionPoints

• **extensionPoints**: IPluginConfigExtensionPoints

Configuration options

`inline`

interface-ipluginsettingsconfig

Interface: IPluginSettingsConfig

Options to initialize the plugin in the settings page.

Hierarchy

- **IPluginSettingsConfig**

Index

Properties

- iframeHeight

Properties

iframeHeight

• **iframeHeight**: number

Height for your plugin iframe

interface-ipoint

Interface: IPoint

Hierarchy

- **IPoint**

Index

Properties

- x
- y

Properties

x

• **x**: number

---

y

• **y**: number

interface-irect

Interface: IRect

Hierarchy

- **IRect**

Index

Properties

- height
- width
- x
- y

Properties

height

• **height**: number

width

• **width**: number

x

• **x**: number

y

• **y**: number

interface-iwidgetmenuitem

Interface: IWidgetMenuItem

Deprecated
This method is deprecated.

`deprecated`

Hierarchy

- **IWidgetMenuItem**

Index

Properties

- onClick
- svgIcon
- tooltip

Properties

onClick

• **onClick**: () => void

svgIcon

• **svgIcon**: string

tooltip

- **tooltip**: string

namespace-sdk

Namespace: SDK

Here you will find a full reference to all publicly available methods in
the Miro SDK for web plugins development.

The Root interface contains the SDK main commands and is a good starting point.

Index

Styling Enumerations

Account and User Interfaces

Board Manipulation Interfaces

Extension Points Interfaces

- BottomBarButton
- DraggableItemsContainerOptions
- ExportMenuButton
- IBoardUICommands
- IPluginConfigExtensionPoints
- ToolbarButton

General Interfaces

- Event
- IBounds
- IOffset
- IPluginConfig
- IPluginSettingsConfig
- IPoint
- IRect
- Root

Other Interfaces

- IWidgetMenuItem

Styling Interfaces

- IEnums

Widgets Manipulation Interfaces

- IBoardTagsCommands
- IBoardWidgetsCommands
- ICardWidget
- IComment
- ICurveWidget
- IDocumentWidget
- IEmbedWidget
- IEmojiWidget
- IFrameWidget
- IImageWidget
- ILineWidget
- IMockupWidget
- IPreviewWidget
- IShapeWidget
- IStickerWidget
- ITag
- ITextWidget
- IWebScreenshotWidget

- IWidget
- IWidgetNamespaces
- IWidgetShortData

Account and User Type aliases

- AccountPermission
- BoardPermission

Extension Points Type aliases

- ButtonExtensionPoint
- UIPanel

General Type aliases

- EventType

Styling Type aliases

- BackgroundColorStyle
- BackgroundOpacityStyle
- BoldStyle
- BorderColorStyle
- BorderOpacityStyle
- BorderWidthStyle
- FontSizeStyle
- HighlightingStyle
- ItalicStyle
- LineColorStyle
- LineThicknessStyle
- PaddingStyle
- StrikeStyle
- TextColorStyle
- UnderlineStyle

Widgets Manipulation Type aliases

- CreateTagRequest
- InputTags
- InputWidget
- InputWidgets
- OneOrMany
- UpdateTagRequest
- WidgetCapabilities
- WidgetMetadata
- WidgetNamespacesKeys
- WidgetToBeCreated

Account and User Type aliases

AccountPermission

Ƭ **AccountPermission**: "MANAGE_APPS"

BoardPermission

Ƭ **BoardPermission**: "EDIT_INFO" | "EDIT_CONTENT" | "EDIT_COMMENTS"

Extension Points Type aliases

ButtonExtensionPoint

Ƭ **ButtonExtensionPoint**: T | () => Promise<T | void>

Type parameters:

Name
`T`

UIPanel

Ƭ **UIPanel**: "toolbar" | "top" | "bottomBar" | "map"

The available panels in the board.

General Type aliases

EventType

Ƭ **EventType**: "SELECTION_UPDATED" | "WIDGETS_CREATED" | "WIDGETS_DELETED" | "WIDGETS_TRANSFORMATION_UPDATED" | "ESC_PRESSED" | "ALL_WIDGETS_LOADED" | "COMMENT_CREATED" | "CANVAS_CLICKED" | "DATA_BROADCASTED" | "RUNTIME_STATE_UPDATED" | "METADATA_CHANGED" | "ONLINE_USERS_CHANGED"

Constant defining different events that can be triggered by the main application.

The following events are experimental and can be unstable:

- ESC_PRESSED
- ALL_WIDGETS_LOADED
- COMMENT_CREATED
- CANVAS_CLICKED
- DATA_BROADCASTED
- RUNTIME_STATE_UPDATED
- METADATA_CHANGED
- ONLINE_USERS_CHANGED

Styling Type aliases

BackgroundColorStyle

Ƭ **BackgroundColorStyle**: string | number

BackgroundOpacityStyle

Ƭ **BackgroundOpacityStyle**: number

BoldStyle

Ƭ **BoldStyle**: 0 | 1

BorderColorStyle

Ƭ **BorderColorStyle**: string | number

BorderOpacityStyle

Ƭ **BorderOpacityStyle**: number

BorderWidthStyle

Ƭ **BorderWidthStyle**: number

FontSizeStyle

Ƭ **FontSizeStyle**: number

HighlightingStyle

Ƭ **HighlightingStyle**: string | number

ItalicStyle

Ƭ **ItalicStyle**: 0 | 1

LineColorStyle

Ƭ **LineColorStyle**: string | number

LineThicknessStyle

Ƭ **LineThicknessStyle**: number

PaddingStyle

Ƭ **PaddingStyle**: 0 | 8

StrikeStyle

Ƭ **StrikeStyle**: 0 | 1

TextColorStyle

Ƭ **TextColorStyle**: string | number

UnderlineStyle

Ƭ **UnderlineStyle**: 0 | 1

Widgets Manipulation Type aliases

CreateTagRequest

Ƭ **CreateTagRequest**: { color: number | string ; title: string ; widgetIds?: InputWidgets }

Commands to interact and manipulate tags

Type declaration:

Name	Type
`color`	number | string
`title`	string
`widgetIds?`	InputWidgets

InputTags

Ƭ **InputTags**: string | string[] | { id: string } | { id: string }[]

Commands to interact and manipulate tags

InputWidget

Ƭ **InputWidget**: string | { id: string }

Representation of one widget by id

InputWidgets

Ƭ **InputWidgets**: string | string[] | { id: string } | { id: string }[]

Representation of one or multiple widgets by id

OneOrMany

Ƭ **OneOrMany**: T | T[]

Type parameters:

Name
`T`

UpdateTagRequest

Ƭ **UpdateTagRequest**: { color?: number | string ; id: string ; title?: undefined | string ; widgetIds?: InputWidgets }

Commands to interact and manipulate tags

Type declaration:

Name	Type
`color?`	number | string
`id`	string
`title?`	undefined | string
`widgetIds?`	InputWidgets

WidgetCapabilities

Ƭ **WidgetCapabilities**: { editable: boolean }

Type declaration:

Name	Type
`editable`	boolean

WidgetMetadata

Ƭ **WidgetMetadata**: { [x:string]: any; }

WidgetNamespacesKeys

Ƭ **WidgetNamespacesKeys**: keyof IWidgetNamespaces

WidgetToBeCreated

Ƭ **WidgetToBeCreated**: { [index:string]: any; type: string }

> **Note**: when creating a widget that requires a URL, if the URL has special characters. Make sure the URL is encoded properly.

Type declaration:

Name	Type
`type`	string

The `window.miro` Object

Interface: Root

This is the same `window.miro` Object and will be your main entry point to the SDK methods.

Hierarchy

- Root

Index

Properties

Methods

Properties

account

• account: IAccountCommands

Contains commands (functions) to access to the Account (Team)
where the Web-plugin was installed

Note: this is not the current user

board

- **board**: IBoardCommands

The current board the user is watching.
Contains commands (functions) to access to the board information.

Available only when the Web-plugin runs in a board

currentUser

- **currentUser**: ICurrentUserCommands

The current user.
Contains commands (functions) to access to the current user information.

enums

- **enums**: IEnums

Contains constants like events and style that can be used with other SDK methods

Methods

__getRuntimeState

▸ **__getRuntimeState**<T>(): Promise<T>

Gets a previously set state from any container extension point (iframe) from your plugin.

You can set this state with __setRuntimeState

Note: This state is not persisted between boards or page reloads.

`experimental`

Type parameters:

Name	Default
`T`	any

Returns: Promise<T>

A promise resolving into the stored state.

__setRuntimeState

▸ **__setRuntimeState**<T>(`data` : T): Promise<T>

Saves a state (any) that can will be accessible across all
the container extension points (iframes) from your plugin.

You can retrieve this state with __getRuntimeState

Note: This state is not persisted between boards or page reloads.

`experimental`

Type parameters:

Name	Default
`T`	any

Parameters:

Name	Type
`data`	T

Returns: Promise<T>

A promise resolving into the previously saved state. Defaults to an empty object.

---

addListener

▸ **addListener**(`event` : EventType, `listener` : (e: Event) => void): void

Subscribe to an event in the board.
Go to EventType to see a list of possible events.

Note: Available only when the Web plugin runs on a board page.

Parameters:

Name	Type	Description
`event`	EventType	A string with an event as defined in EventType
`listener`	(e: Event) => void	A function to handle the Event

Returns: void

void

Example

JavaScript

```
function handleSelectionUpdated(event) {
  // Your event handler
}
miro.addListener('SELECTION_UPDATED', handleSelectionUpdated)
```

Related: See removeListener to remove a listener

broadcastData

▸ **broadcastData**(`data` : any): void

Broadcast some data to iframes from your plugin in Container Entry Points

You can subscribe to the `DATA_BROADCASTED` event to receive this data.
See addListener and EventType

Parameters:

Name	Type	Description
`data`	any	The payload you wish to broadcast

Returns: void

void

Example

JavaScript

```
miro.broadcastData({
 value: 1
})
```

getClientId

▸ **getClientId**(): string

Returns the clientId from the web plugin.
This is the same clientId from the App settings dashboard.

Returns: string

The web plugin clientId.

getIdToken

▸ **getIdToken**(): Promise<string>

The `getIdToken()` method returns a JSON Web Token (JWT), an encoded form of JSON data, signed by the application secret. You can use JWTs to authenticate requests from your Miro web-plugin to your backend services. You can use an existing JWT library to decode the token and establish the communication between your Miro web-plugin and your backend services.

Returns: Promise<string>

a JWT token

Example

JavaScript

```
miro.getIdToken().then((jwt) =>{
  console.log('jwt token', jwt);
})
```

initialize

▸ **initialize**(`config?` : IPluginConfig): Promise<void>

Accepts a configuration to initialize the plugin.

Available only on a board. Not available in iframes from Container Entry points

Example

JavaScript

```
miro.onReady(() => {
  miro.initialize({
    //plugin configuration
  });
}
```

Parameters:

Name	Type
`config?`	IPluginConfig

Returns: Promise<void>

initializeSettings

▸ **initializeSettings**(`config?` : IPluginSettingsConfig): Promise<void>

Accepts a configuration to initialize the plugin in the settings page.
Available only in the setting page.

Example

JavaScript

```
miro.onReady(() => {
  miro.initializeSettings({
    //plugin configuration
  });
}
```

Parameters:

Name	Type
`config?`	IPluginSettingsConfig

Returns: Promise<void>

isAuthorized

‣ **isAuthorized**(): Promise<boolean>

Check if the current user has authorized the Web-plugin to make API requests on their behalf

Returns: Promise<boolean>

True if the web plugin is authorized, false when not.

Example

JavaScript

```javascript
miro.isAuthorized().then( (isAuthorized) => {
  if (isAuthorized) {
    console.log('Web plugin authorized');
  } else {
    console.log('Unauthorized');
  }
})
```

onReady

‣ **onReady**(`callback` : () => void): void

Callback executed when everything is loaded and ready to use SDK methods.

Parameters:

Name	Type	Description
`callback`	() => void	Function to be executed

Returns: void

Example

JavaScript

```javascript
miro.onReady(() => {
 console.log("Ready to call SDK methods")
}
```

removeListener

▸ **removeListener**(`event` : EventType, `listener` : (e: Event) => void): void

Unsubscribe from an event in the board.
Go to EventType to see a list of possible events.

Note: Available only when the Web plugin runs on a board page.

Parameters:

Name	Type	Description
`event`	EventType	A string with an event as defined in EventType
`listener`	(e: Event) => void	The function originally passed in addListener to handle the event

Returns: void

void

Example

JavaScript

```javascript
miro.removeListener('SELECTION_UPDATED', handleSelectionUpdated)
```

Related: See addListener to add a listener

requestAuthorization

▸ **requestAuthorization**(`options?` : AuthorizationOptions): Promise<void>

Opens a modal to follow the authorization process for your App.

Parameters:

Name	Type	Description
`options?`	AuthorizationOptions	Options for the authorization process. See AuthorizationOptions

Returns: Promise<void>

A promise fulfilled if the app has been authorized.

showErrorNotification

▸ **showErrorNotification**(`text` : string): Promise<void>

Similar to showNotification with additional style to indicate error

Parameters:

Name	Type	Description
`text`	string	The text to show in the notification

Returns: Promise<void>

A Promise, fulfilled when the notification is displayed to the user
Example

JavaScript

```
miro.showErrorNotification('This is an error notification')
```

showNotification

▸ **showNotification**(`text` : string): Promise<void>

Shows a notification to the user

Parameters:

Name	Type	Description
`text`	string	The text to show in the notification

Returns: Promise<void>

A Promise, fulfilled when the notification is displayed to the user

Example

JavaScript

```
miro.showNotification('Hello world')
```

Resources

Changelog

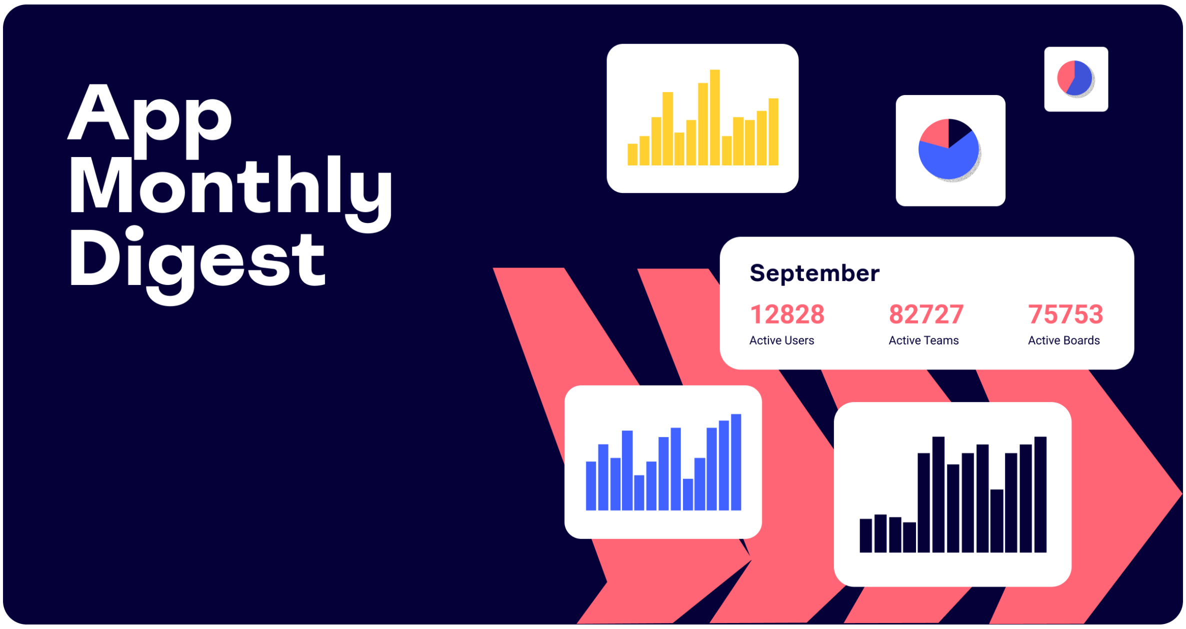
To provide the best experience, we improve the Miro platform almost every week by releasing new features, squashing bugs, and delivering improved documentation. If you're curious about what's new and what's changed in the Miro Platform, you're in the right place. Here's an account of what's new and what we've gradually rolled out:

September

Web SDK

- We've announced the deprecation of the `getToken` method and introduced the `getIdToken` method in March 2021. The `getToken` method stands decommissioned as of September 29, 2021.
- We've announced the deprecation of the `miro.authorize` method and introduced the `miro.requestAuthorization` method in March 2021. The `miro.authorize` method stands decommissioned as of September 29, 2021.

App analytics



We've made the following enhancements to our monthly app statistics mailer:

- We've added monthly app stats. What's more, if you want to evaluate what's been happening in the past 3 months or look at historical app stats, you can easily view the numbers by simply scrolling down.
- We've simplified the language to make our content more user-friendly. It's all about numbers and graphs—everybody loves graphs!
- We've enhanced the structure of the content in the email to make it more consistent with our different mailers.

Documentation

- We've enhanced the REST API Authorization topic to include step-by-step instructions, with detailed field descriptions and examples.

August

REST API

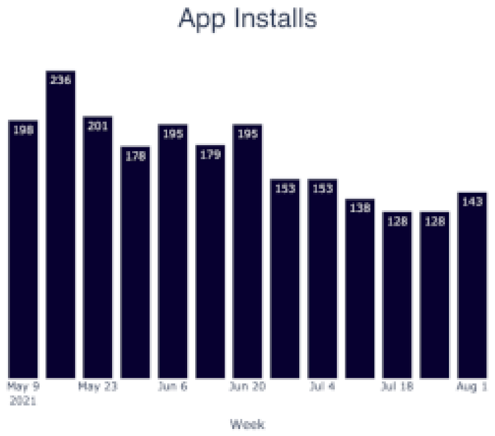
- Expire user authorization token: Using an expiring access token and refresh token enhances your application's security. An access token expires in 1 hour and a refresh token expires in 60 days. When a new access token is requested, you also get a new refresh token.

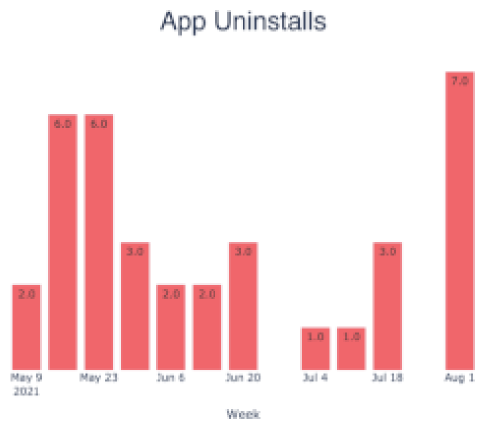
Documentation

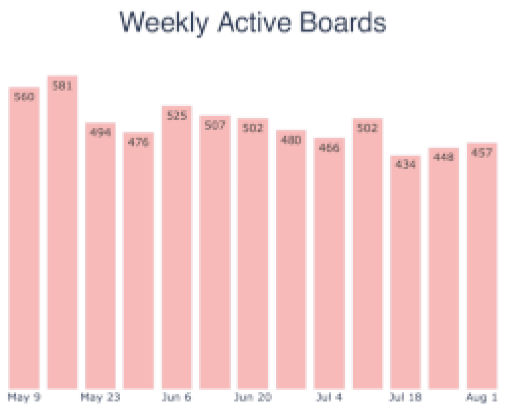
- We’ve updated the Design guidelines section. To provide a better user experience for your users, it’s a good idea to use consistent UI and app components with the rest of the Miro product UI. While we are working on an official set of reusable components, we recommend using Mirotone CSS components—a base library that enables everyone to design and build their Miro apps.
- We’ve completely overhauled our Web SDK Getting Started Guide. Join us as we take you through the steps to build your first “Hello, World” Miro app.

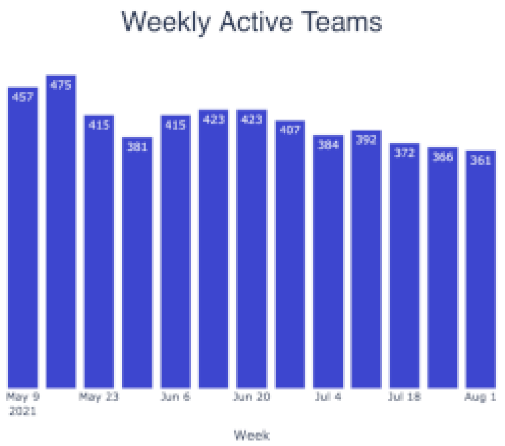
App analytics

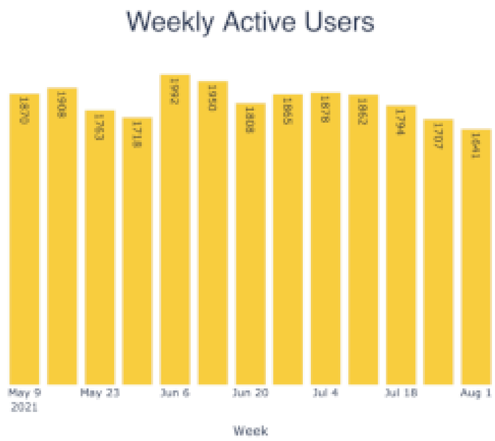
- If you have an app in the marketplace, you can now look at your app metrics via your monthly Miro Marketplace app report. This report will be in your inbox on the first Wednesday of every month. Knowing your app metrics enables you to understand the health and performance of your application, allowing you to make informed, data-driven decisions.







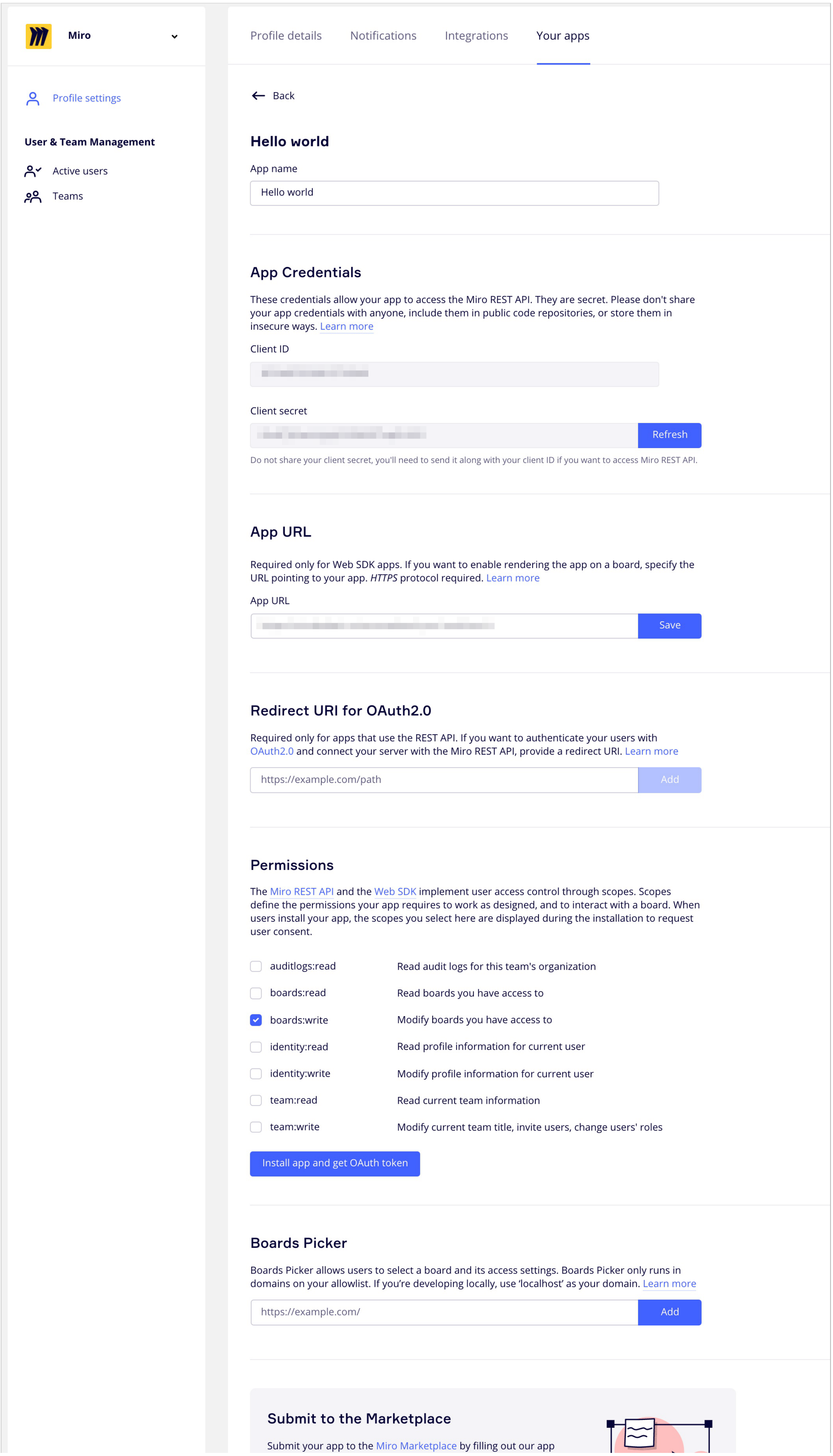




June

- boardsPicker Component Web SDK: The `miroBoardsPicker.open` method response now returns a new URL format within the `access-link` field. The new URL format is `https://miro.com/app/live-embed/{board_id}?boardAccessToken={token}&autoplay=true`.
  The new URL is now based on the direct-link live embed URL and you can use the URL parameters per your requirement. For more information, see boardsPicker Component.

- User interface: We've enhanced the user interface and content of the App settings page for a better developer experience when creating, submitting, and maintaining your apps.



Hello world

App name

Hello world

App Credentials

These credentials allow your app to access the Miro REST API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways. Learn more

Client ID

[redacted]

Client secret

[redacted]

Do not share your client secret, you'll need to send it along with your client ID if you want to access Miro REST API.

App URL

Required only for Web SDK apps. If you want to enable rendering the app on a board, specify the URL pointing to your app. *HTTPS* protocol required. Learn more

App URL

[redacted]

Redirect URI for OAuth2.0

Required only for apps that use the REST API. If you want to authenticate your users with OAuth2.0 and connect your server with the Miro REST API, provide a redirect URI. Learn more

https://example.com/path

Permissions

The Miro REST API and the Web SDK implement user access control through scopes. Scopes define the permissions your app requires to work as designed, and to interact with a board. When users install your app, the scopes you select here are displayed during the installation to request user consent.

□ auditlogs:read	Read audit logs for this team's organization
□ boards:read	Read boards you have access to
☑ boards:write	Modify boards you have access to
□ identity:read	Read profile information for current user
□ identity:write	Modify profile information for current user
□ team:read	Read current team information
□ team:write	Modify current team title, invite users, change users' roles

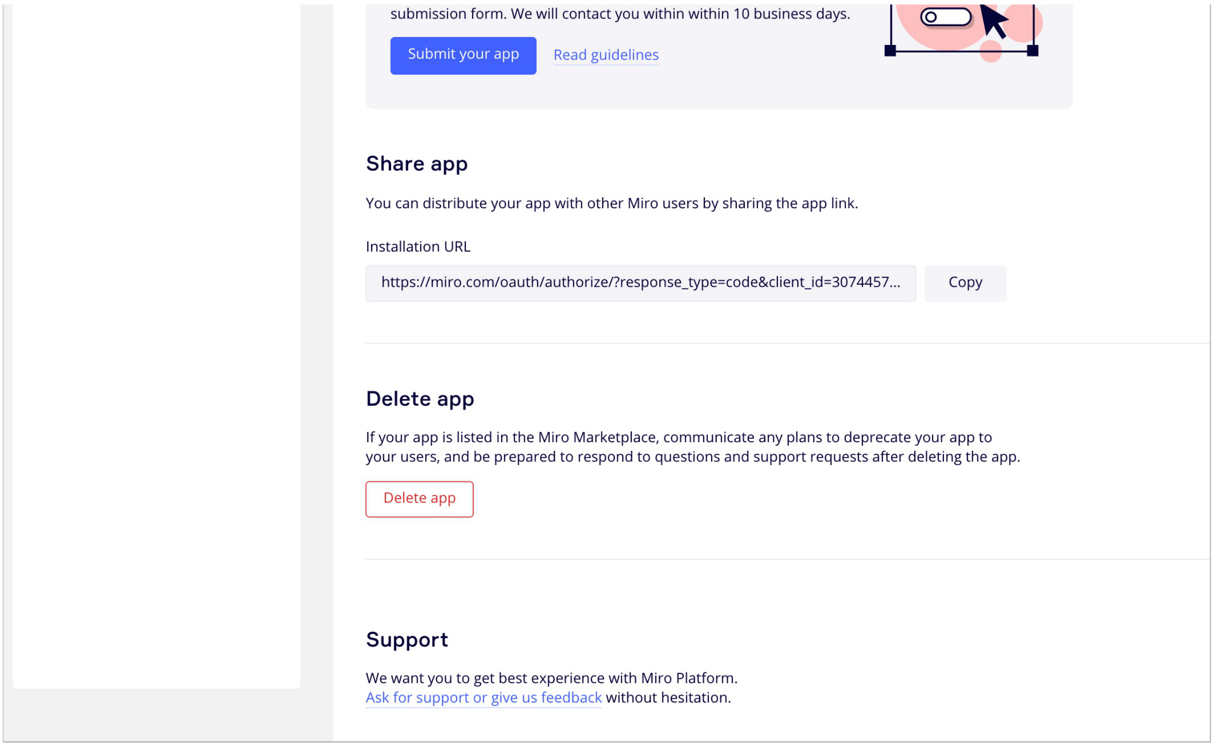
Boards Picker

Boards Picker allows users to select a board and its access settings. Boards Picker only runs in domains on your allowlist. If you're developing locally, use 'localhost' as your domain. Learn more

https://example.com/

Submit to the Marketplace

Submit your app to the Miro Marketplace by filling out our app



submission form. We will contact you within within 10 business days.

Share app

You can distribute your app with other Miro users by sharing the app link.

Installation URL

https://miro.com/oauth/authorize/?response_type=code&client_id=3074457...

Delete app

If your app is listed in the Miro Marketplace, communicate any plans to deprecate your app to your users, and be prepared to respond to questions and support requests after deleting the app.

Support

We want you to get best experience with Miro Platform.
Ask for support or give us feedback without hesitation.

# April

- REST API and Web SDK: To make it easier to align text of different sizes, we've updated the default value for the `padding` style property of the Text widget to `0`. For more information on the SDK, see interface-itextwidget. For more information on the REST API, see Text.

- REST API and Web SDK: We've deprecated the following Text widget style properties: `borderColor`, `borderWidth`, `borderStyle`, `borderOpacity`, and `padding`. For more information on the SDK, see interface-itextwidget. For more information on the REST API, see Text.

March

- Web SDK: We've deprecated the `getToken` method and introduced the `getIdToken` method.

- Web SDK: We've deprecated the `miro.authorize` method and introduced the `miro.requestAuthorization` method.

Support and community

- Join our Developer community on Discord to exchange ideas and to chat with other Miro developers.

- Stuck somewhere? Encountered any blockers or errors? Need assistance? We're here to help you! Ask your question on our developer community forum.

Policies

Developer terms of use

App development policy

Security: Public apps must follow Miro security guidelines

User Experience: Every application must provide the best user experience. Applications and developers are prohibited from:

- Degrading or compromising the performance of Miro services.
- Using vulgar or obscene language or images. Your application must not contain or offer content that is violent, extreme, or inappropriate.
- Offering sexually-oriented or adult content. Your application must not contain or offer content that a reasonable person considers pornographic or indecent.
- Displaying inappropriate communications through your application. Examples of inappropriate communications include, but are not limited to, hate speech, shaming, and messages that promote harmful or illegal behavior.

Business: While using Miro Developer Platform APIs and SDKs, developers must agree to respect our business. Every application must behave in accordance with appropriate and accepted business conduct. As part of good business practices, applications and developers are prohibited from:

- Circumventing Miro’s intended branding or limitations including, but not limited to, pricing, features, and access structures. You cannot use Miro APIs or SDKs to replicate or compete with Miro’s core products or services.
- Advertising, including display ads, within the application experience. In addition, applications cannot use data or content from Miro in any advertisements, or for purposes of targeting advertisements or contacting users.

Design: Good design is an important part of product development. We want all users to enjoy a delightful experience. We support developers in their efforts to build applications that provide meaningful and relevant user experiences. Please provide your users with excellent, well-designed products. As part of good design practices, applications and developers are prohibited from:

- Infringing on any intellectual property rights in your design. Your application is well-designed, high-quality, distinctive, and doesn’t misuse the Miro brand.
- Changing the application’s look, feel, function, operation, or disclosures after Miro review. Any changes to these elements must be submitted again for review.

Use of Data: Protecting user data, statistics, analytics and other information (collectively, “Data”) is paramount at Miro and it must be for you too. You are responsible for good data stewardship practices. First and foremost, you have no independent rights to any Data. Your applications must not store any user's personal data and always retrieve current user Data at the time of use via Miro APIs.

In accordance with this, applications and developers are prohibited from:

- Collecting, storing, and using Data without obtaining proper consent of the user.
- Using Data to contact users. If you want to contact users outside of Miro, you must gain permission through a clear and separate permissions process. You can only contact users for emergencies in which the safety and security of the user is at risk and in compliance with the law.

- Asking users to provide sensitive, private, and confidential personal information, such as credit card numbers or passwords, unless specifically necessary as part of the application's legitimate function and purpose.
- Renting, selling, or sharing Data with third parties under any circumstances.
- Creating applications that encourage installers to circumvent or interfere with their own workplace and employer data, or with privacy and security policies.
- Ignoring a user's request for deletion. When a user deletes your application, or if you discontinue your application, you must delete all associated Data within 14 business days.
- Combining Data with data gathered from other sources for any purposes unrelated to the use of the application.
- Requesting and using scopes, also known as permissions, not required for your application's functioning. Use only the appropriate and necessary scopes, and clearly define the need for scopes within your application's description.
- Failing to notify users about privacy and their Data. Your application must include a publicly-available and easily-accessible privacy policy that complies with applicable laws and that explains how the application collects, processes, and stores personal data. Your privacy policy must include, among other things, information about data storage, security controls, data retention, and individual rights.
- Accessing Data for surveillance purposes. You cannot allow or assist any entity to conduct surveillance or obtain Data using your access to the Miro APIs.
- Exploiting Data in a way not approved by Miro and not disclosed to and permitted by users.

Law and Safety: Applications should not create unsafe environments or hardships for users. Each application must comply with all applicable laws and legal requirements in all locations where it is made available to users. In addition, applications and developers are prohibited from:

- Spamming, harassing, stalking, intimidating, or threatening users.
- Allowing impersonation of users or otherwise allowing for false representations within the application.
- Facilitating violations of the law.
- Infringing on anyone's intellectual property rights.
- Representing that your application is authorized by or produced by another company or organization.
- Allowing or facilitating financial transactions conducted in an insecure and unapproved manner.
- Permitting use of your application by children under the age of 16.

Additional Requirements: Applications and developers must follow this policy, as well as other applicable guidelines and policies, including the Developer Terms of Use and Security Guidelines.

Data Breach: If Data is breached, exposed, or otherwise compromised through your application, you must immediately inform Miro at privacy@miro.com.

Deprecation policy

On occasion, Miro may choose to retire a service offered in the Miro Developer Platform. This may be because the future roadmap for the service requires breaking changes (thus demanding a new major version), or because the service depends on, or relates to, user-facing product capabilities that are being retired.

As the retirement of a service is a breaking change for applications that depend on it, Miro will provide a minimum of 6 months' notice before retiring a service. This 6 month period will begin with a "deprecation announcement" that communicates our intention to retire a service on a given future date. This announcement will be made through our communication channels. We will also attempt to notify the developers of affected apps directly by email on a best effort basis.

The period between a deprecation announcement and the retirement of a service is known as the deprecation period. During the deprecation period of a service:

- No new features or capabilities will be added to the service
- No new beta versions or experimental features will be released
- Requesting a beta version will return the production version
- Features labeled as experimental will remain so until retirement
- Only critical security bugs will be addressed, and only in the production version

Deprecation announcements will be accompanied by guidance on how to respond if your application is impacted by the announcement, including a migration guide if a new major version is available or recommendations of alternative third-party services where possible.

Retirement

At the end of the deprecation period, the service concerned will be retired and no longer be accessible to applications. After this date, any attempt to use the service will fail. All documentation for the service will also be taken offline.