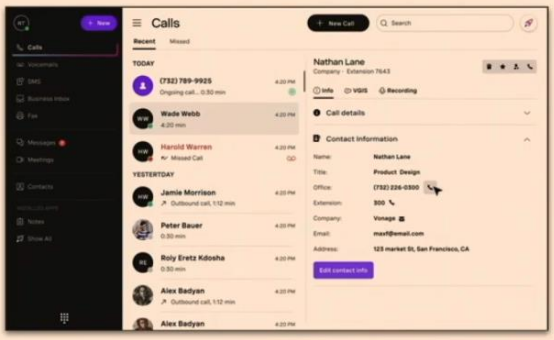
Why is this Relevant

Our Challenges...

About Vonage

Vonage Business Cloud
Vue.js Application

nexmo®
The Vonage® API Platform



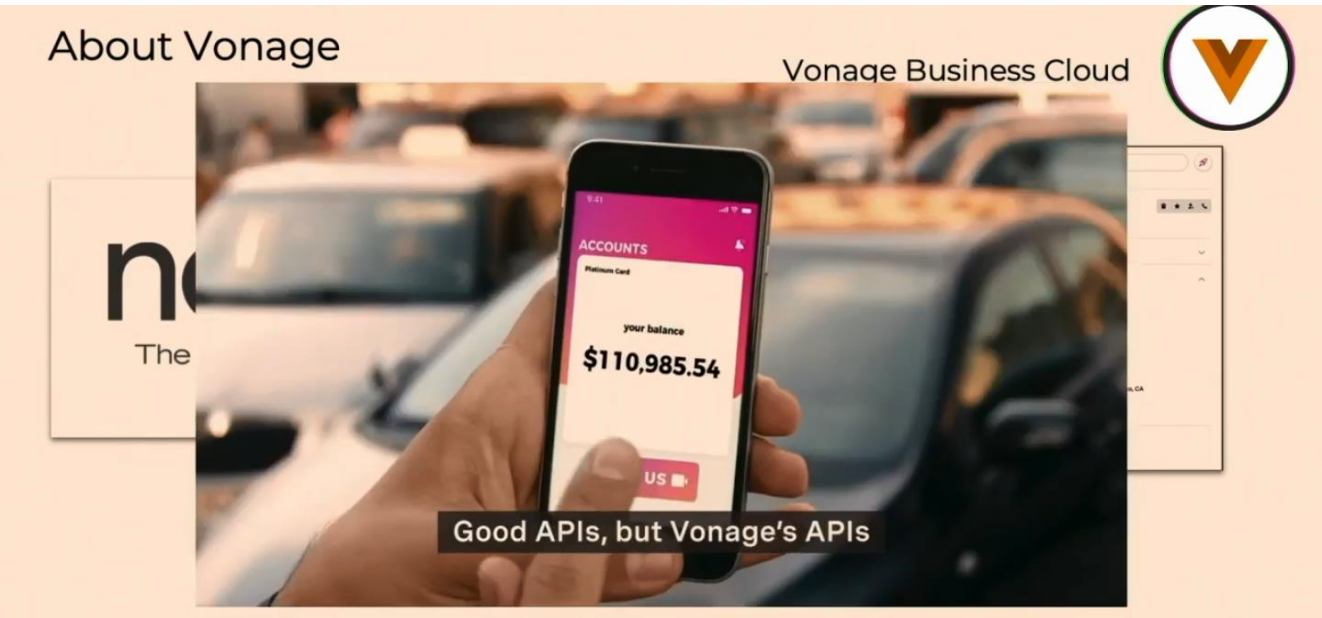
About Vonage

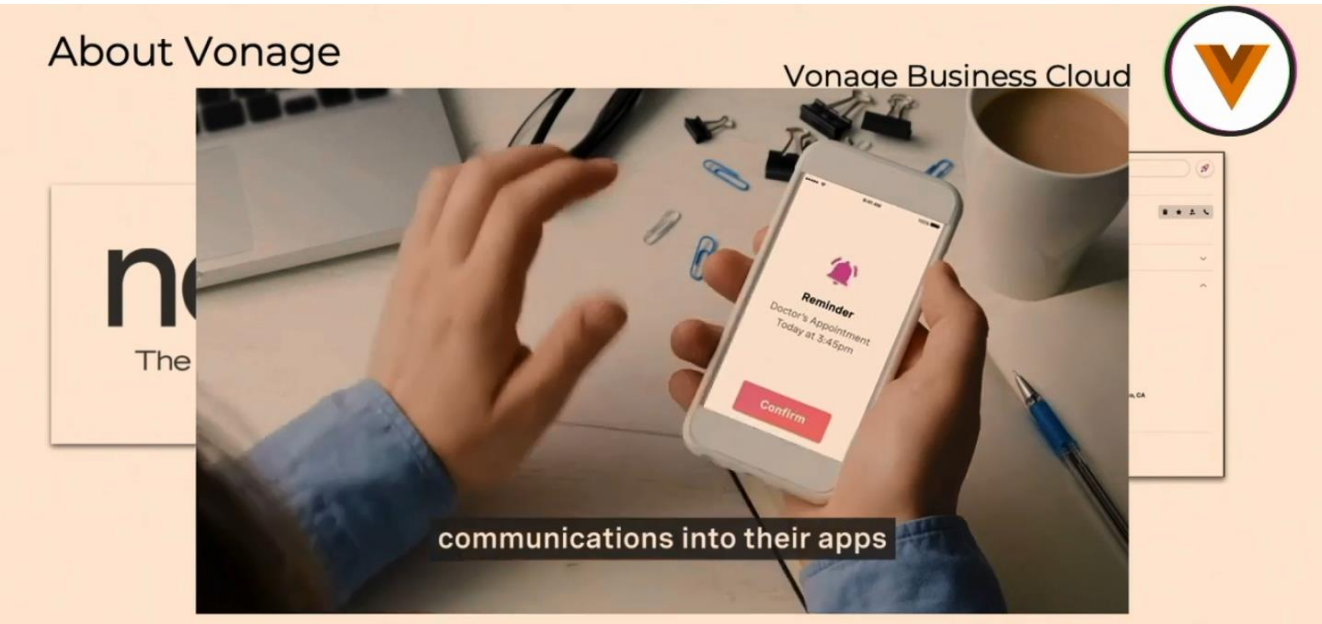
Vonage Business Cloud

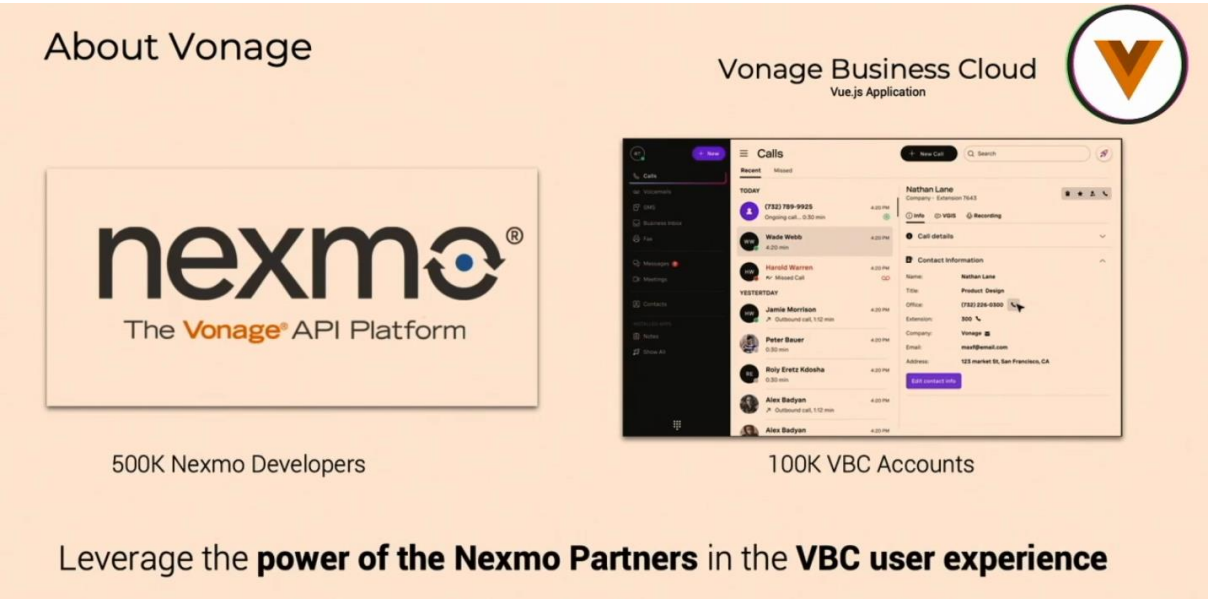
VONAGE
Communications APIs
Unified Communications
Contact Centers

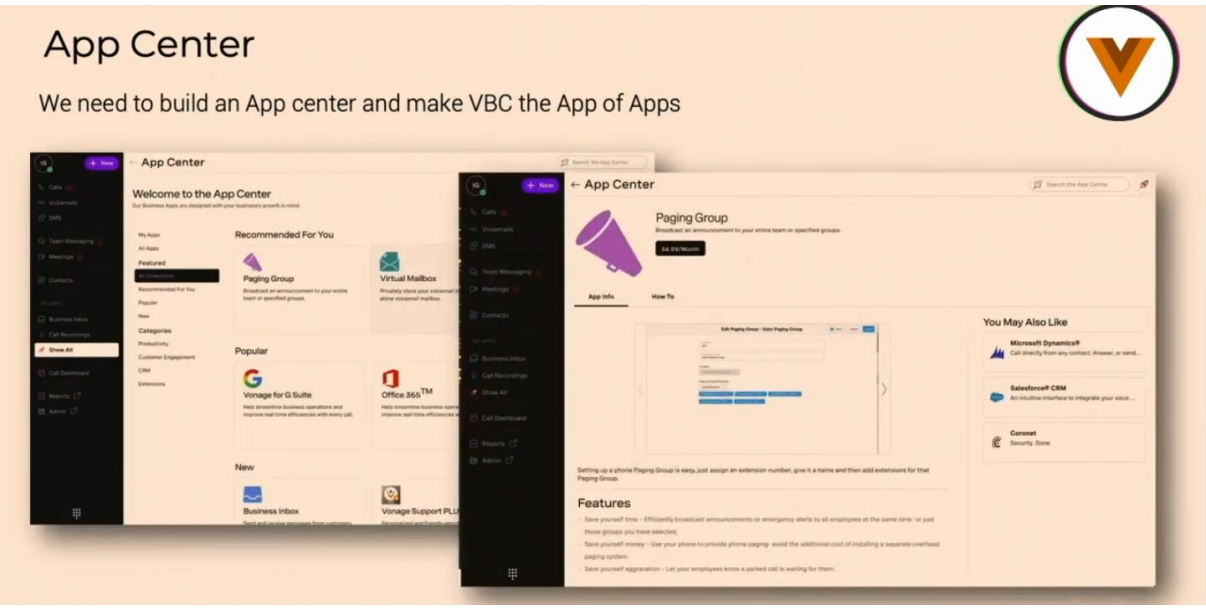
Click to find out more


About Vonage
Vonage Business Cloud
VONAGE
Vonage's Unified Communications

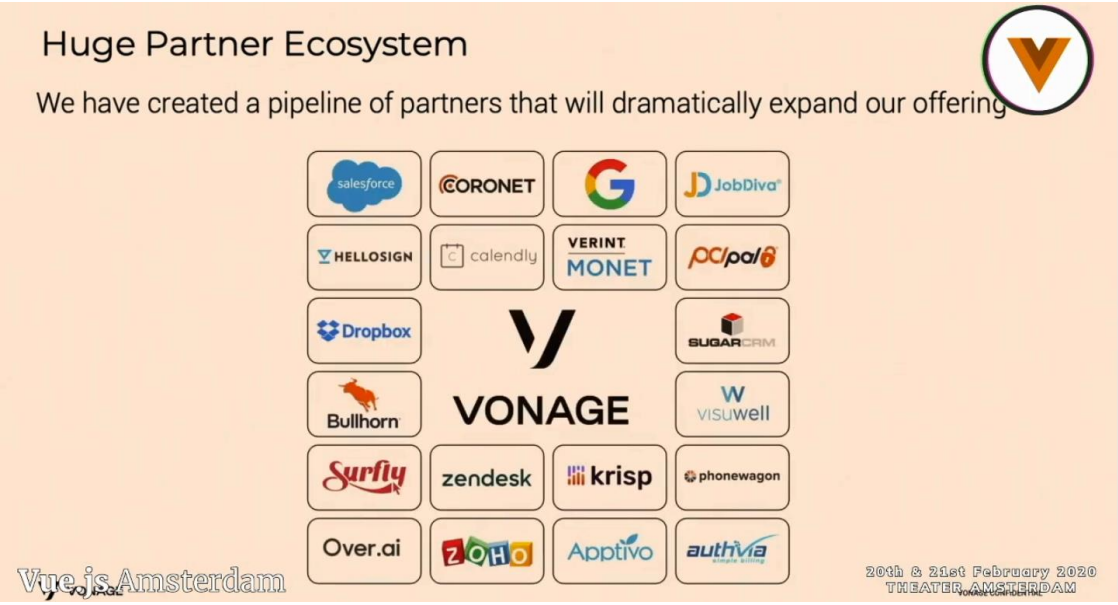

About Vonage
Vonage Business Cloud
ACCOUNTS
your balance
$110,985.54
Good APIs, but Vonage's APIs


About Vonage
Vonage Business Cloud
Reminder
Doctor's Appointment
Today at 3:45pm
Confirm
communications into their apps



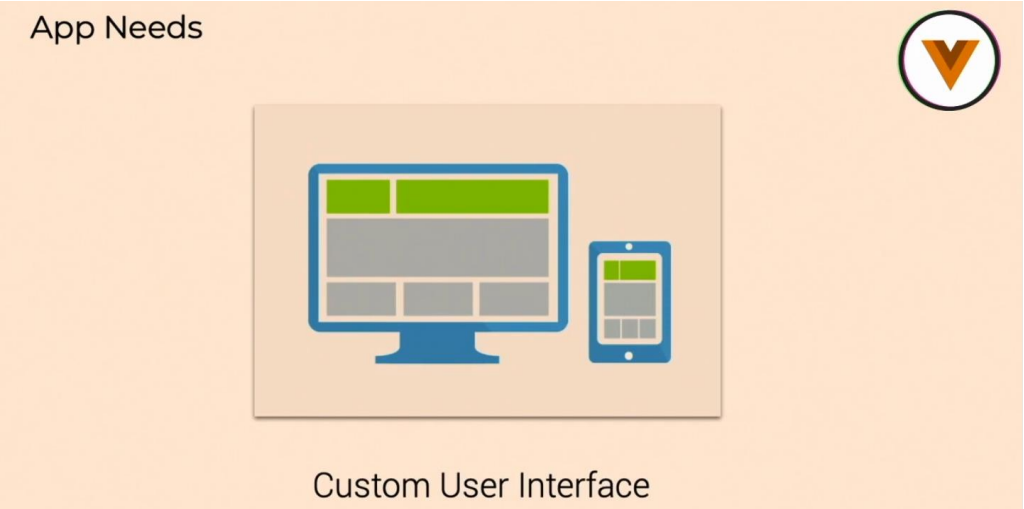


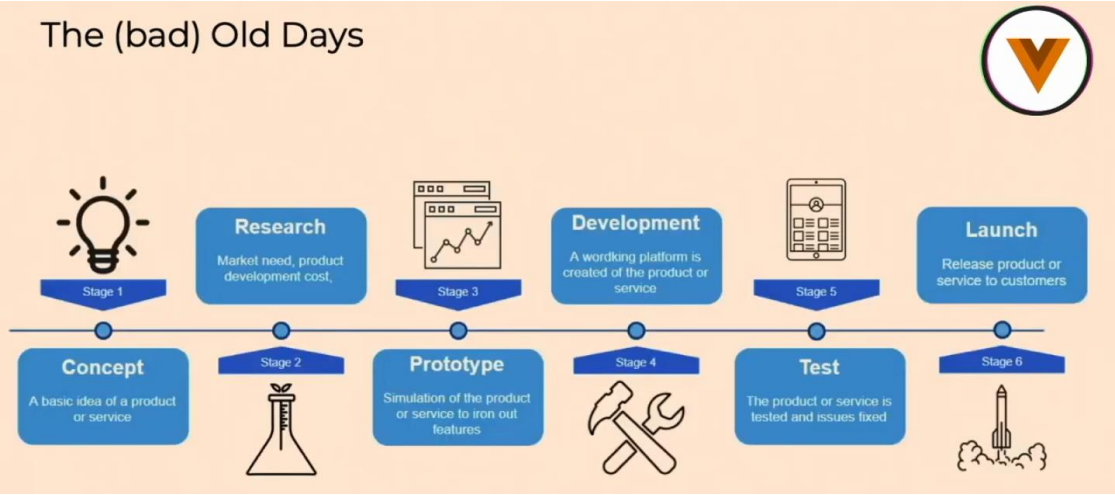
The App Center enables 3rd party developers to integrate their applications into VBC for their customers to use it



What Does an App Need?

We also need to get developers to embed their custom UI into our Vonage app

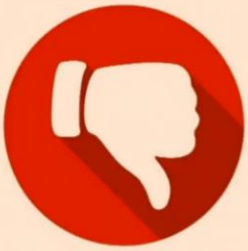




The (bad) Old Days

Is there better way?

Microservices

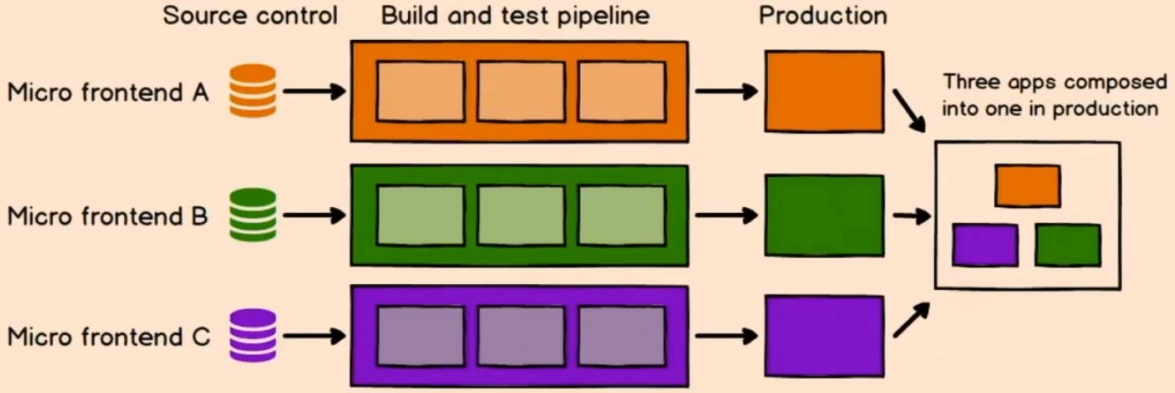


Frontend challenges

- Unified user experience
- Small Bundle size
- Autonomous
- **No standard library**

What are Micro Frontends?

Micro Frontends

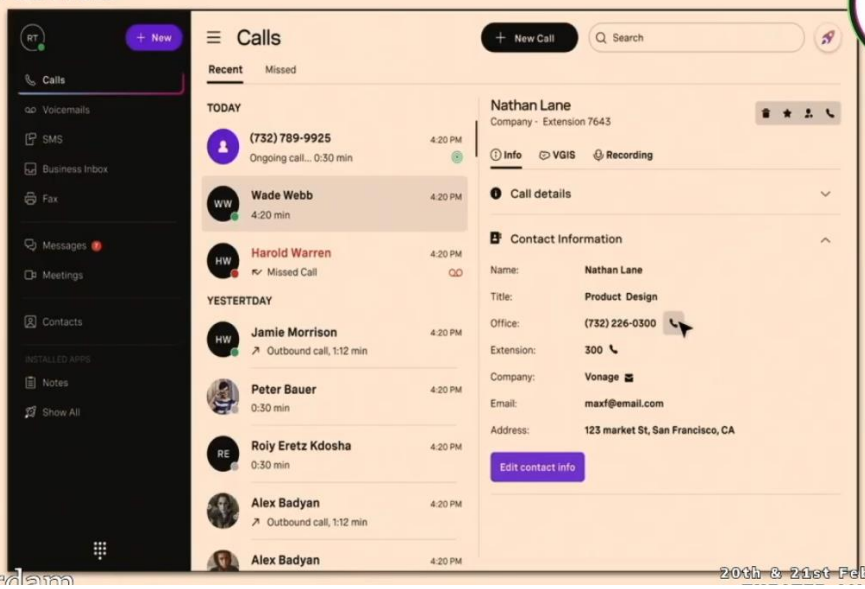


Micro Frontends is Awesome

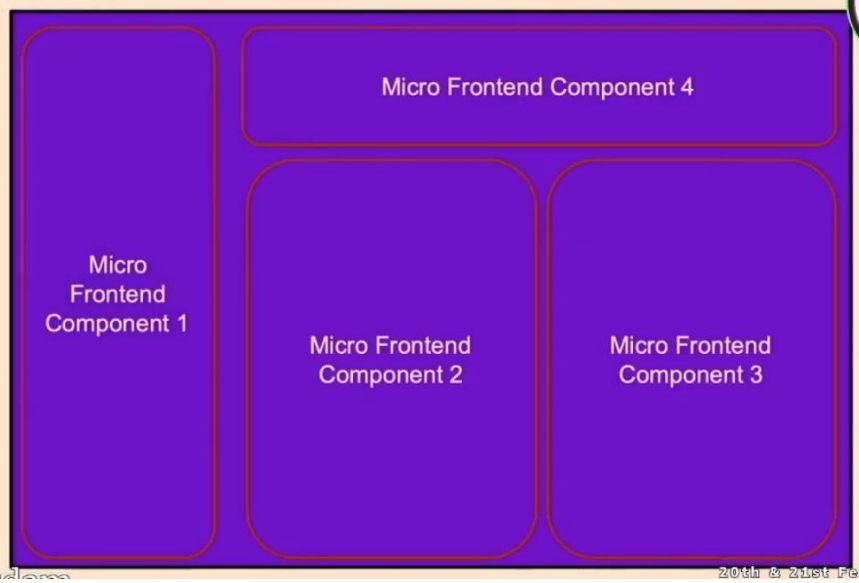
- Maintainability
- Tech Freedom
- Independent Deployment
- Customizable

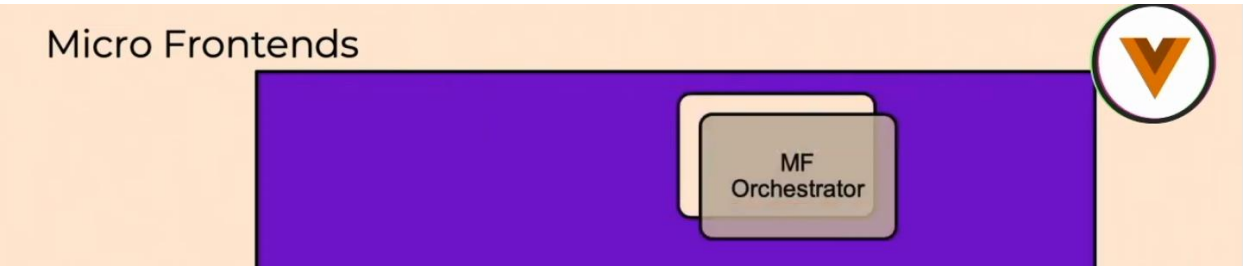
AWESOME

Micro Frontends

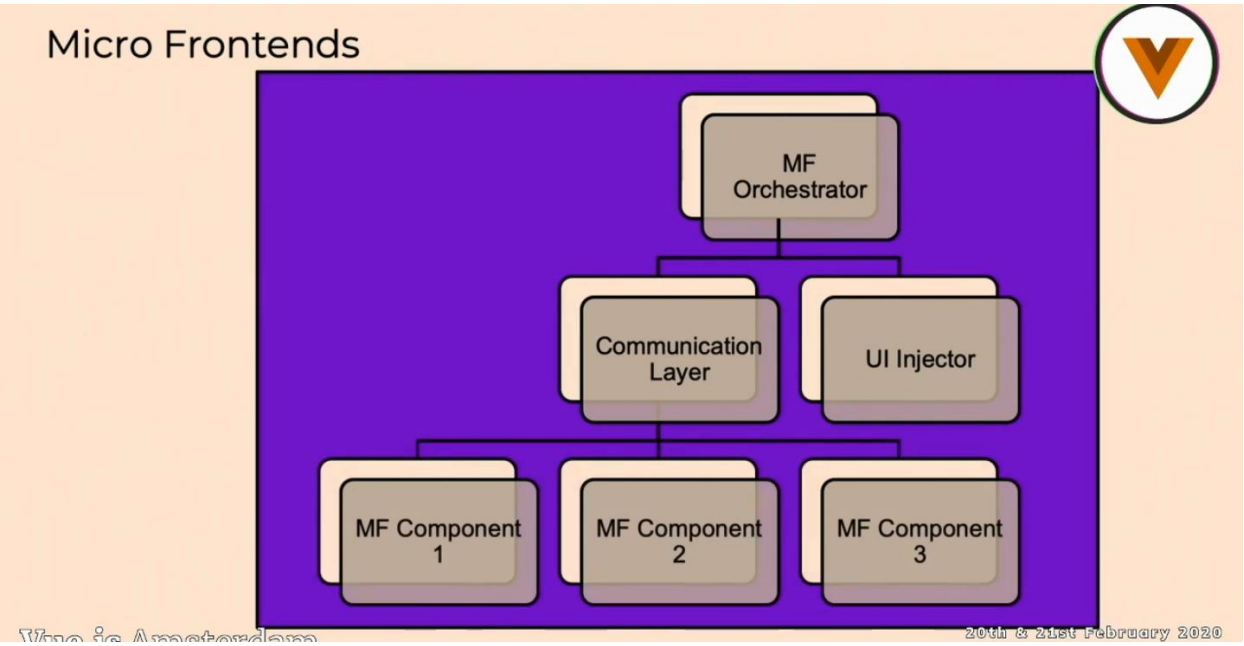


Micro Frontends

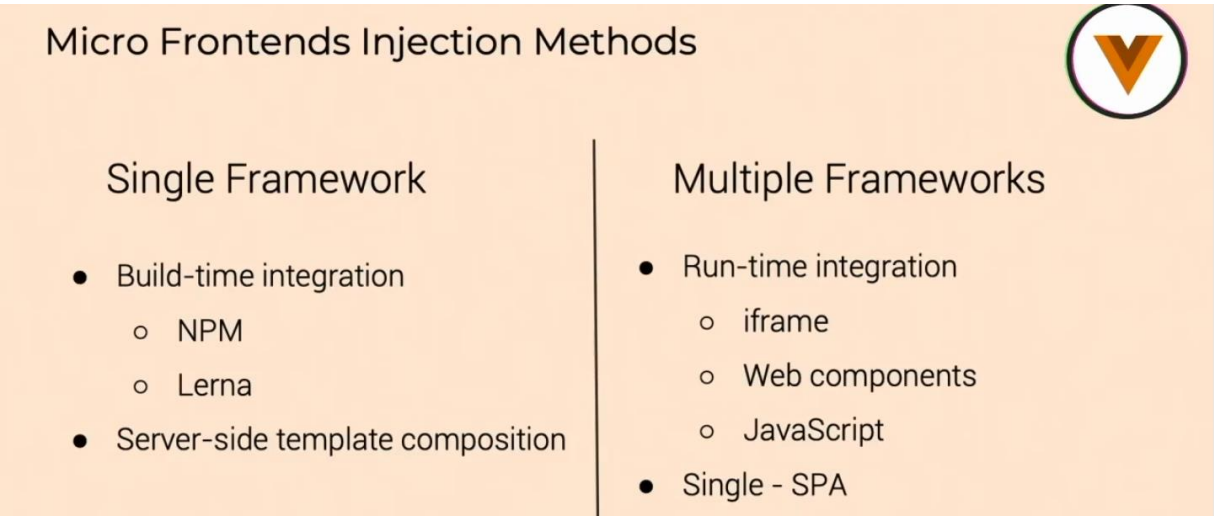




This is a pure JS file that exposes a simple interface, its key function is use an injection configuration and inject micro-frontend apps/component into the screen. It also enables a micro-frontend app to communicate with the parent app.



How to Inject Micro Frontend Component?



Micro Frontends Injection Methods

Build-time integration

- Independent repos ✅
- Easy versioning ✅
- Sharing dependencies ✅
- Used by top brands ✅
- Easiest to test ✅
- Require rebuild for changes ❌
- Not customizable ❌

Micro Frontends Injection Methods

Run-time integration Web-Component

- Technology agnostic ✅
- Shadow DOM ✅
- Not used with many brands ❌

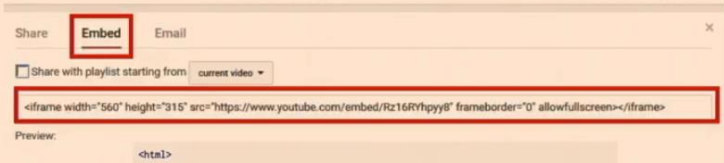
```html
<html>
  <head>
    <meta charset="utf-8" />
    <title>my-component demo</title>
    <script src="https://unpkg.com/vue"></script>
    <script src="./my-component.umd.js"></script>
  </head>

  <body>
    <my-component></my-component>
  </body>
</html>
```

Micro Frontends Injection Methods

Run-time integration Iframes

- Absolut Isolation ✅
- Widely supported ✅
- Easy to integrate ✅
- Used by top brands ✅
- Old fashion ❌
- No shared dependencies ❌




```html
<html>
  <head>
    <title>Feed me!</title>
  </head>
  <body>
    <h1>Welcome to Feed me!</h1>

    <iframe id="micro-frontend-container"></iframe>

    <script type="text/javascript">
      const microFrontendsByRoute = {
        '/': 'https://browse.example.com/index.html',
        '/order-food': 'https://order.example.com/index.html',
        '/user-profile': 'https://profile.example.com/index.html',
      };

      const iframe = document.getElementById('micro-frontend-container');
      iframe.src = microFrontendsByRoute[window.location.pathname];
    </script>
  </body>
</html>
```

Micro Frontends Injection Methods

Run-time Integration JS & Single SPA

- Highly flexible ✅
- Simple to convert Existing applications ✅
- Easily send data ✅
- No isolation ❌

```html
<html>
  <head>
    <title>Feed me!</title>
  </head>
  <body>
    <h1>Welcome to Feed me!</h1>

    <!-- These scripts don't render anything immediately -->
    <!-- Instead they attach entry-point functions to `window` -->
    <script src="https://browse.example.com/bundle.js"></script>
    <script src="https://order.example.com/bundle.js"></script>
    <script src="https://profile.example.com/bundle.js"></script>

    <div id="micro-frontend-root"></div>

    <script type="text/javascript">
      // These global functions are attached to window by the above scripts
      const microFrontendsByRoute = {
        '/': window.renderBrowseRestaurants,
        '/order-food': window.renderOrderFood,
        '/user-profile': window.renderUserProfile,
      };
      const renderFunction = microFrontendsByRoute[window.location.pathname];

      // Having determined the entry-point function, we now call it,
      // giving it the ID of the element where it should render itself
      renderFunction('micro-frontend-root');
    </script>
  </body>
</html>
```

This is about downloading a JS file at runtime that has a render method that gets a selector for what component to render and where to render it at.

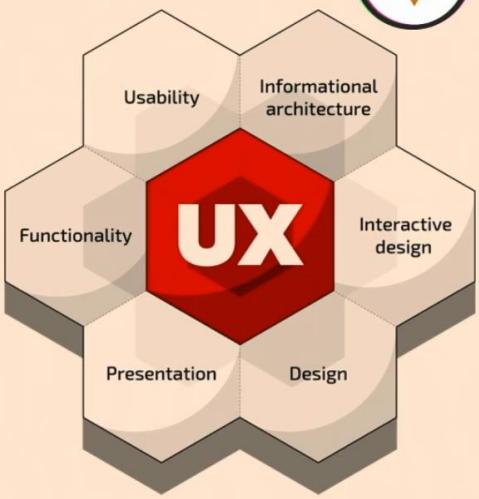
So How does Vonage Injects MF Components?

Micro Frontends Injection Methods

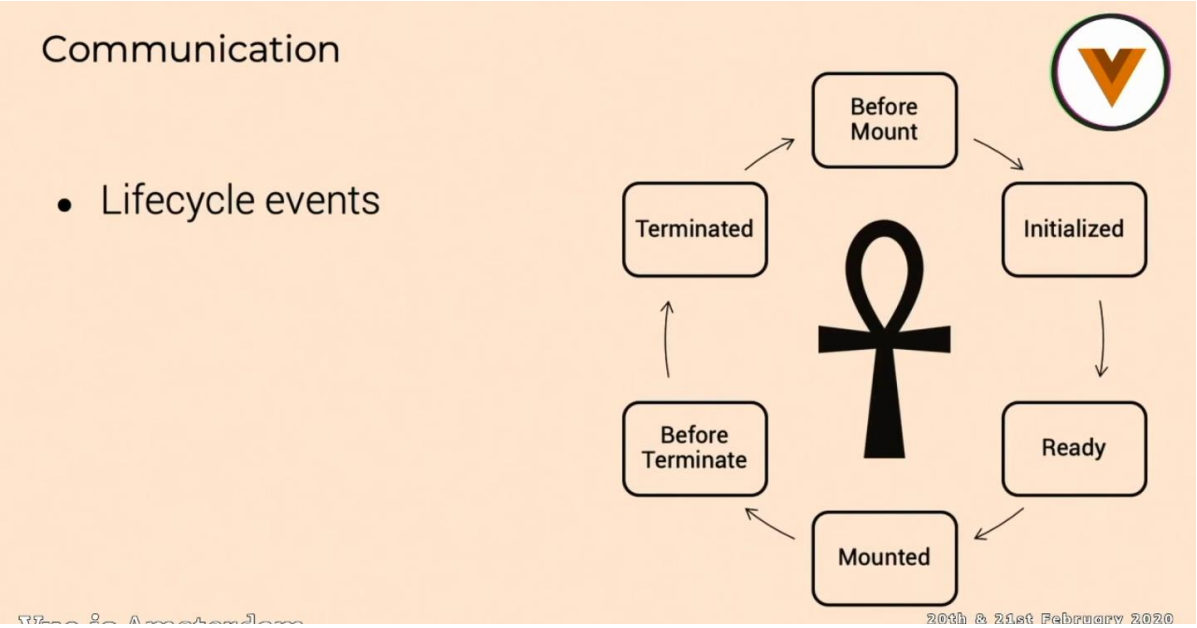
Unified User Experience

Unified User Experience

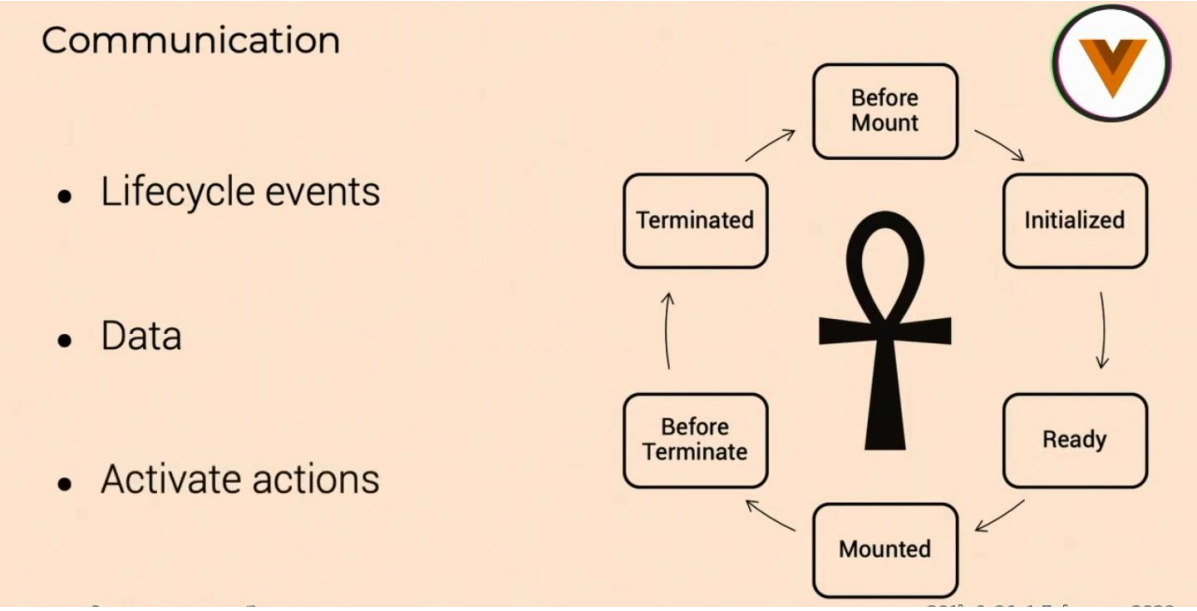
- Shared CSS libraries
- Shared component libraries



What do Apps Talk About?



Every micro-frontend (MFE) we create has to report its **lifecycle events** as they happen to the parent application. An MFE reports an **initialized** event once its downloaded and it's going to start running, it reports **ready** event after calling all its APIs and getting back the data and ready to be presented, the MFO then reports the **mount** event for the MFE.



Data transfer is also possible between the MFEs through the MFO.

Communication

@vonage/micro-frontends

0.1.0-alpha • Public • Published 6 hours ago

Readme | Explore BETA | 3 Dependencies | 0 Dependents

@vonage/micro-frontends

A simple solution for Micro Frontends orchestration.

Use the MicroFrontendOrchestrator in your host application, along with the MicroFrontendComponent in the hosted components. Create your custom communication API between the host and the hosted components, and use the built in lifecycle events. Supported Micro frontend implementations are both Iframe and Web Components (native as well as Vue.js compiled web components).

Installation

```
npm install @vonage/micro-frontends
```

MicroFrontendOrchestrator (MFO)

Use the MFO to inject and manage multiple instances of your micro frontend components.

MicroFrontendComponent (MFC)

Use the MFC in your micro frontend components to report basic lifecycle events, send custom events and subscribe to host custom events.

Usage Example

```
import { MicroFrontendComponent } from '@vonage/micro-frontends';

const mfc = new MicroFrontendComponent();
const initialData = await mfc.initialize(); // get some inital data from the host app
// do some verification on your end...
verifyInitialData(initialData);
// your component is ready to be presented.
await mfc.ready();
// your component is now displayed.
```

API

All functions (except one) return a Promise. All function accept the options object with the following fields:

- requestTimeout (ms) - The promise is rejected if no answer was received from the host applicaiton after this time. Default value is 10 seconds.

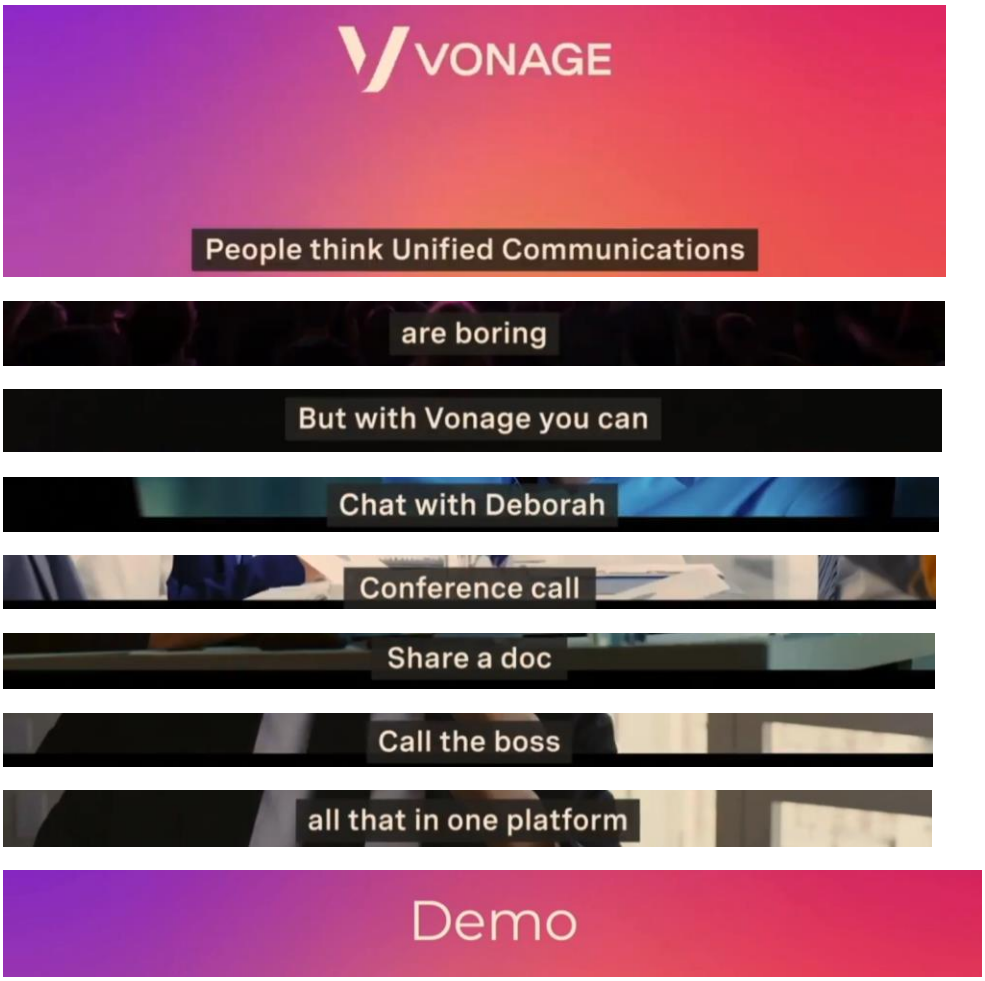
More options soon...

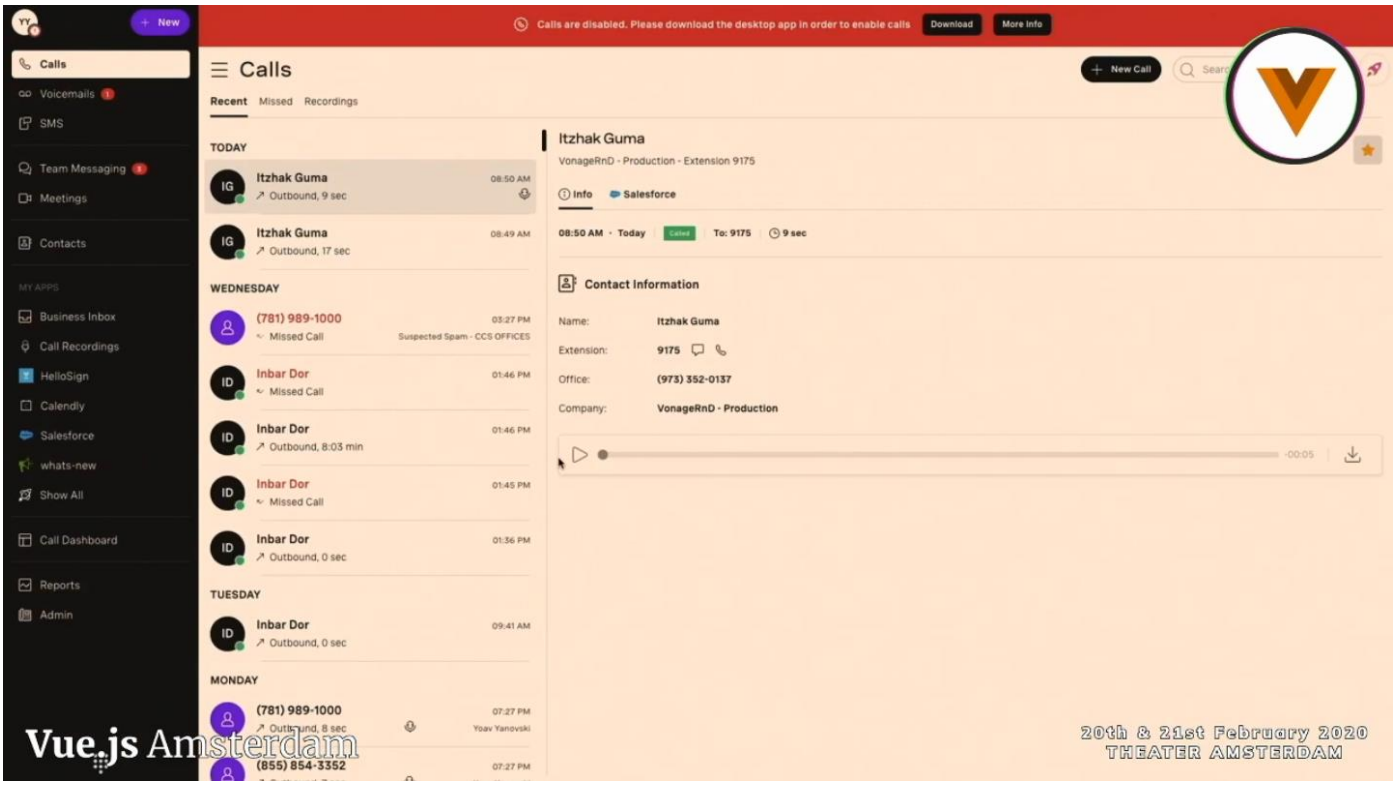
Component Lifecycle

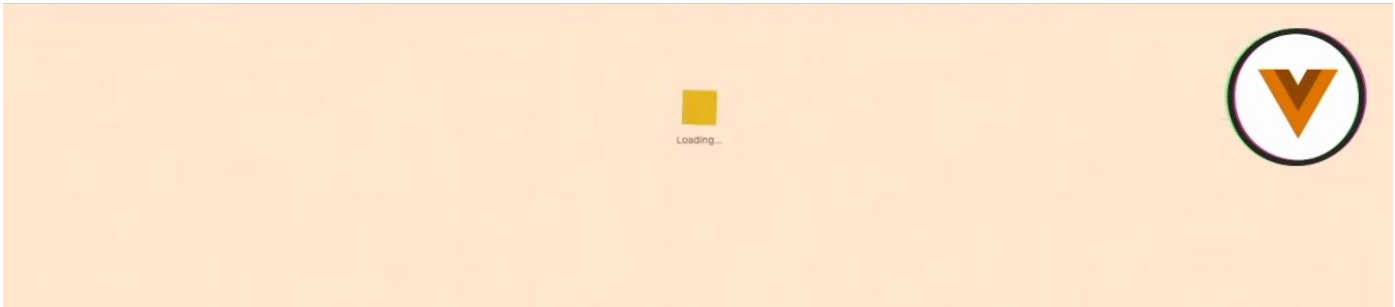
initialize

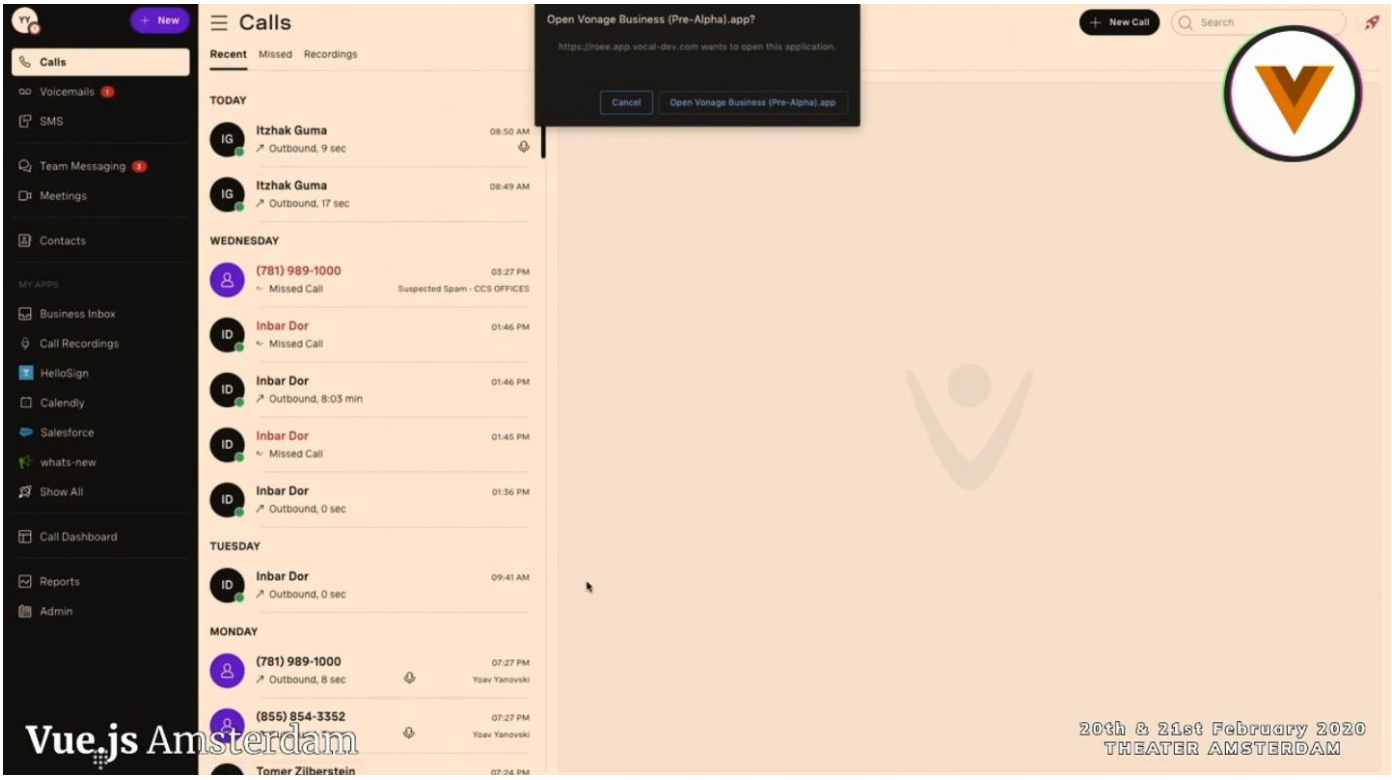
Tell the host application you want to initialize a new component. This method is usually used to pass some initial verification data from the host application, such as a token.

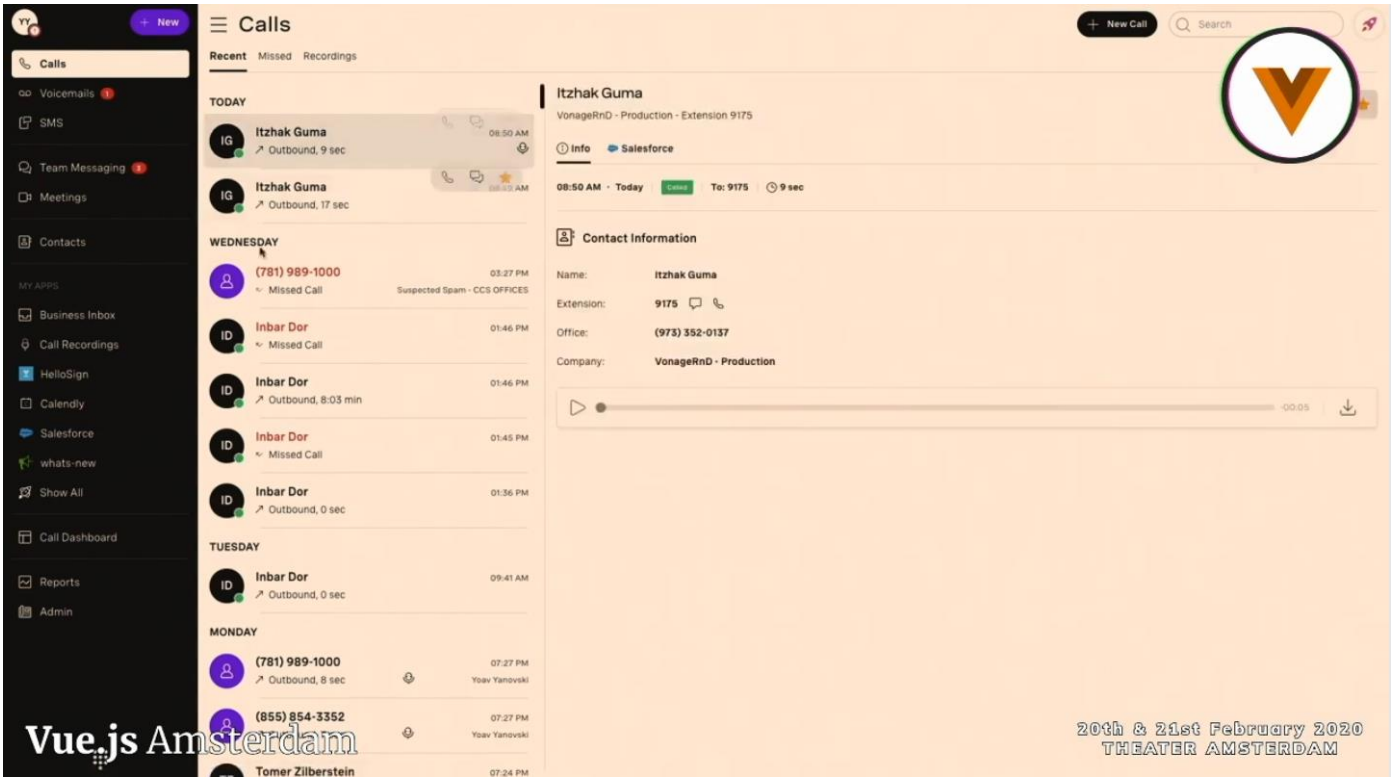
We created the **MFC SDK** (for web and for mobile) that wraps all the communication for the developer via methods/APIs.

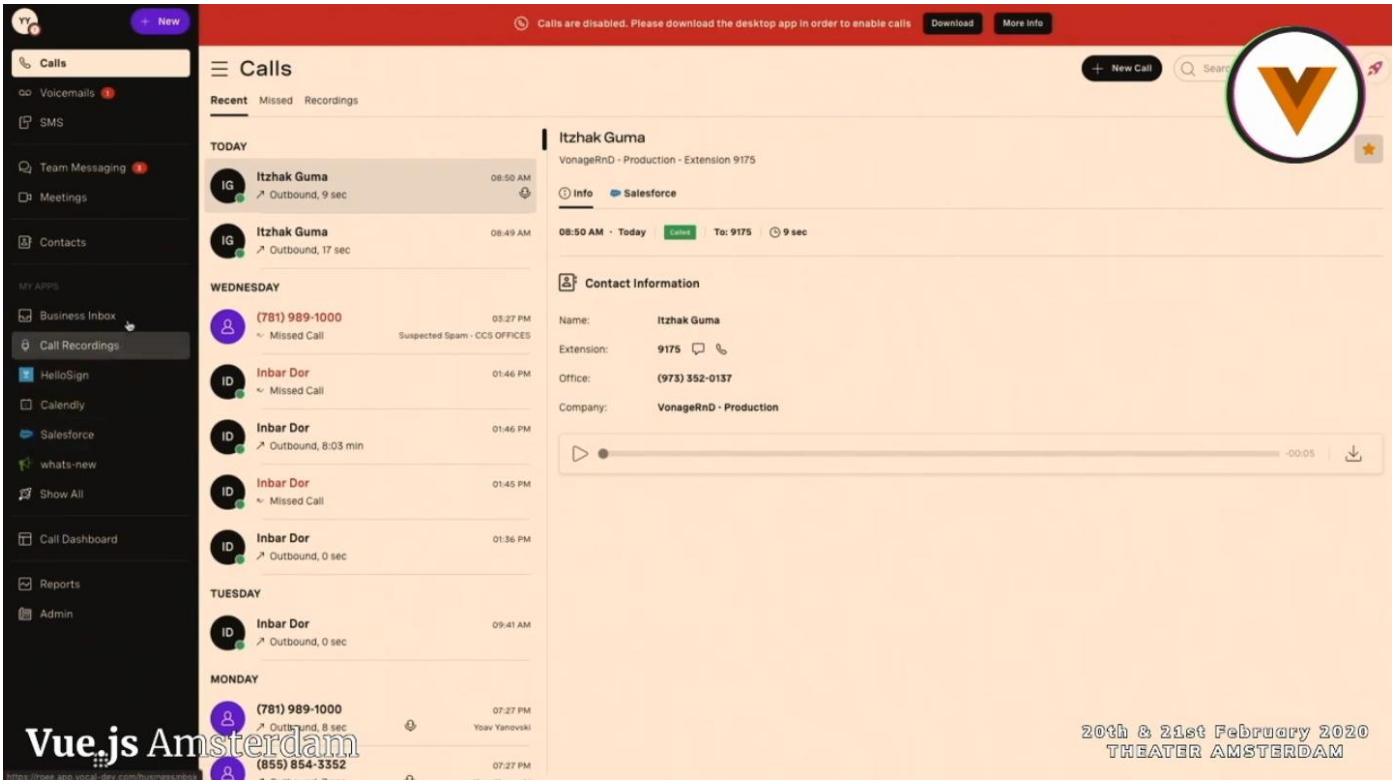


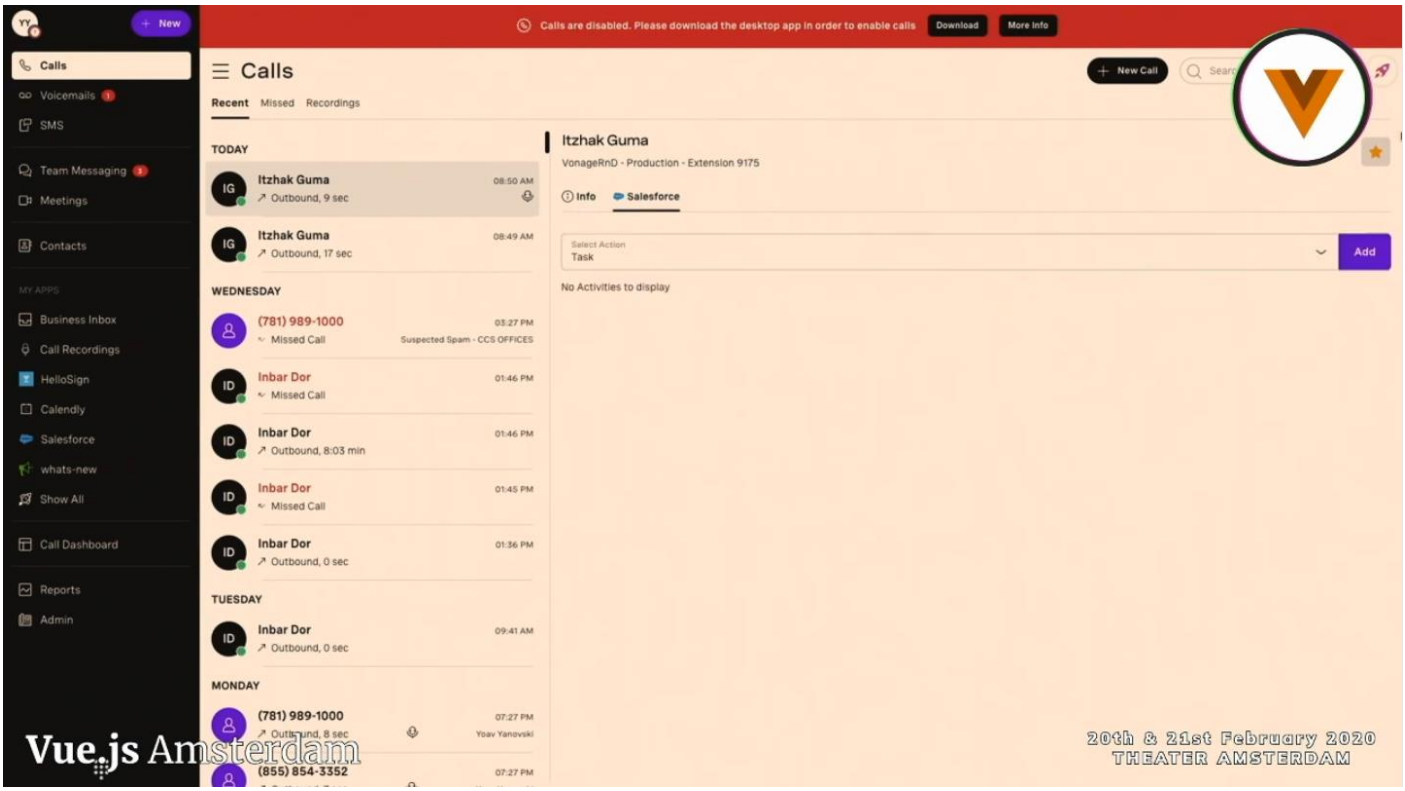

Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
New
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
Recent
Missed
Recordings
New Call
TODAY
Itzhak Guma
08:50 AM
Outbound, 9 sec
Itzhak Guma
08:49 AM
Outbound, 17 sec
WEDNESDAY
(781) 989-1000
03:27 PM
Missed Call
Suspected Spam - CCS OFFICES
Inbar Dor
01:46 PM
Missed Call
Inbar Dor
01:46 PM
Outbound, 8:03 min
Inbar Dor
01:45 PM
Missed Call
Inbar Dor
01:36 PM
Outbound, 0 sec
TUESDAY
Inbar Dor
09:41 AM
Outbound, 0 sec
MONDAY
(781) 989-1000
07:27 PM
Yoav Yanovski
(855) 854-3352
07:27 PM
Itzhak Guma
VonageRnD - Production - Extension 9175
Info
Salesforce
08:50 AM · Today
Called
To: 9175
9 sec
Contact Information
Name:
Itzhak Guma
Extension:
9175
Office:
(973) 352-0137
Company:
VonageRnD - Production
-00:05
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM

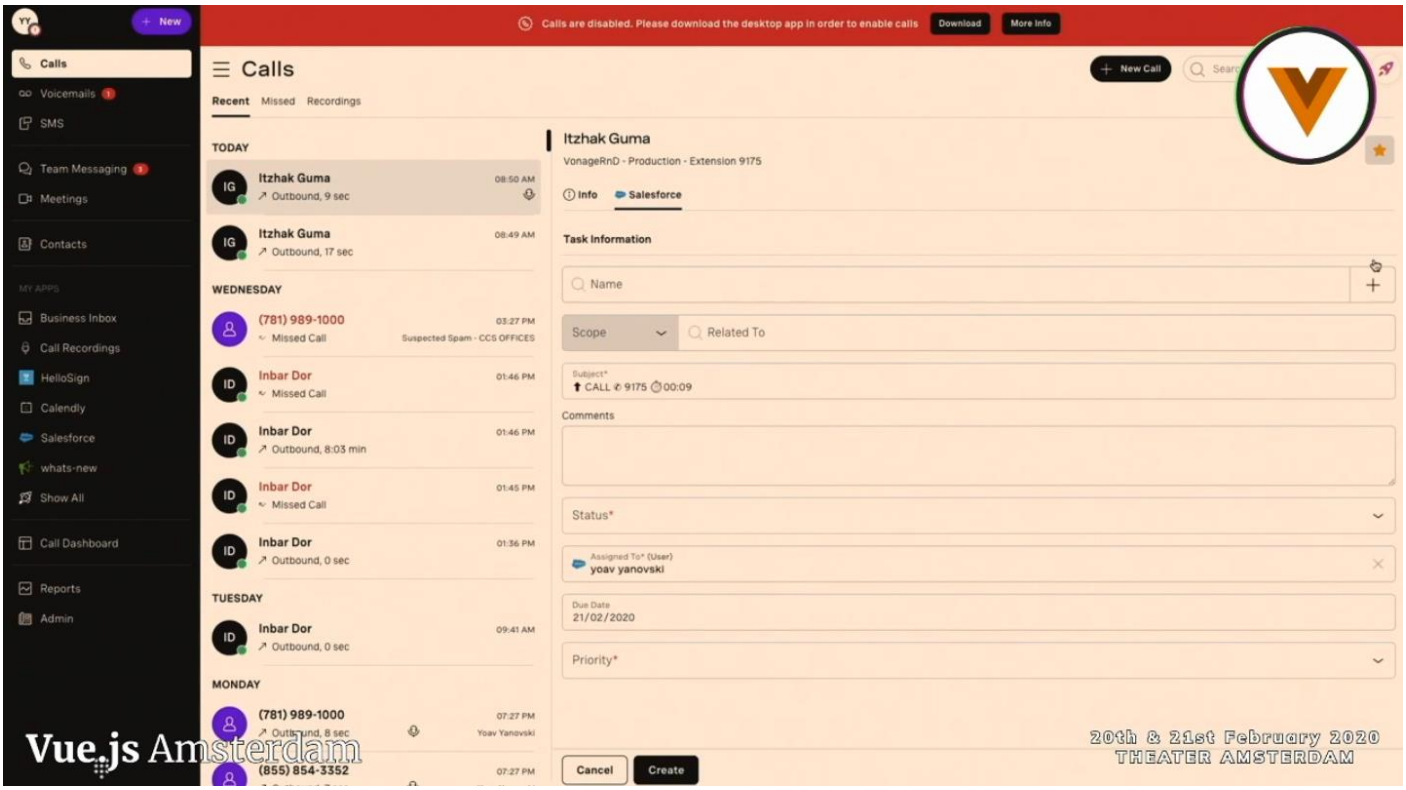

Loading...

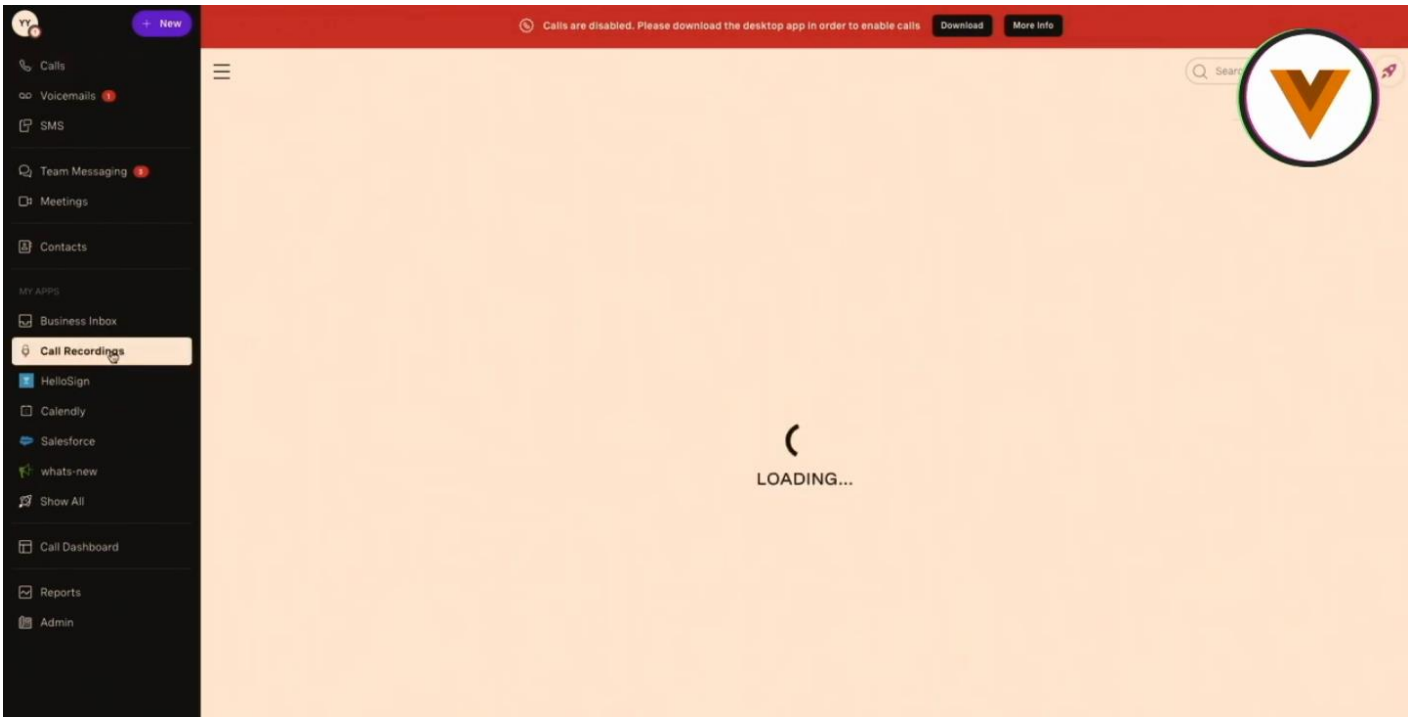

New
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
Recent
Missed
Recordings
Open Vonage Business (Pre-Alpha).app?
https://roee.app.vocal-dev.com wants to open this application.
Cancel
Open Vonage Business (Pre-Alpha).app
New Call
Search
TODAY
Itzhak Guma
08:50 AM
Outbound, 9 sec
Itzhak Guma
08:49 AM
Outbound, 17 sec
WEDNESDAY
(781) 989-1000
03:27 PM
Missed Call
Suspected Spam - CCS OFFICES
Inbar Dor
01:46 PM
Missed Call
Inbar Dor
01:46 PM
Outbound, 8:03 min
Inbar Dor
01:45 PM
Missed Call
Inbar Dor
01:36 PM
Outbound, 0 sec
TUESDAY
Inbar Dor
09:41 AM
Outbound, 0 sec
MONDAY
(781) 989-1000
07:27 PM
Outbound, 8 sec
Yoav Yanovski
(855) 854-3352
07:27 PM
Yoav Yanovski
Tomer Zilberstein
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM

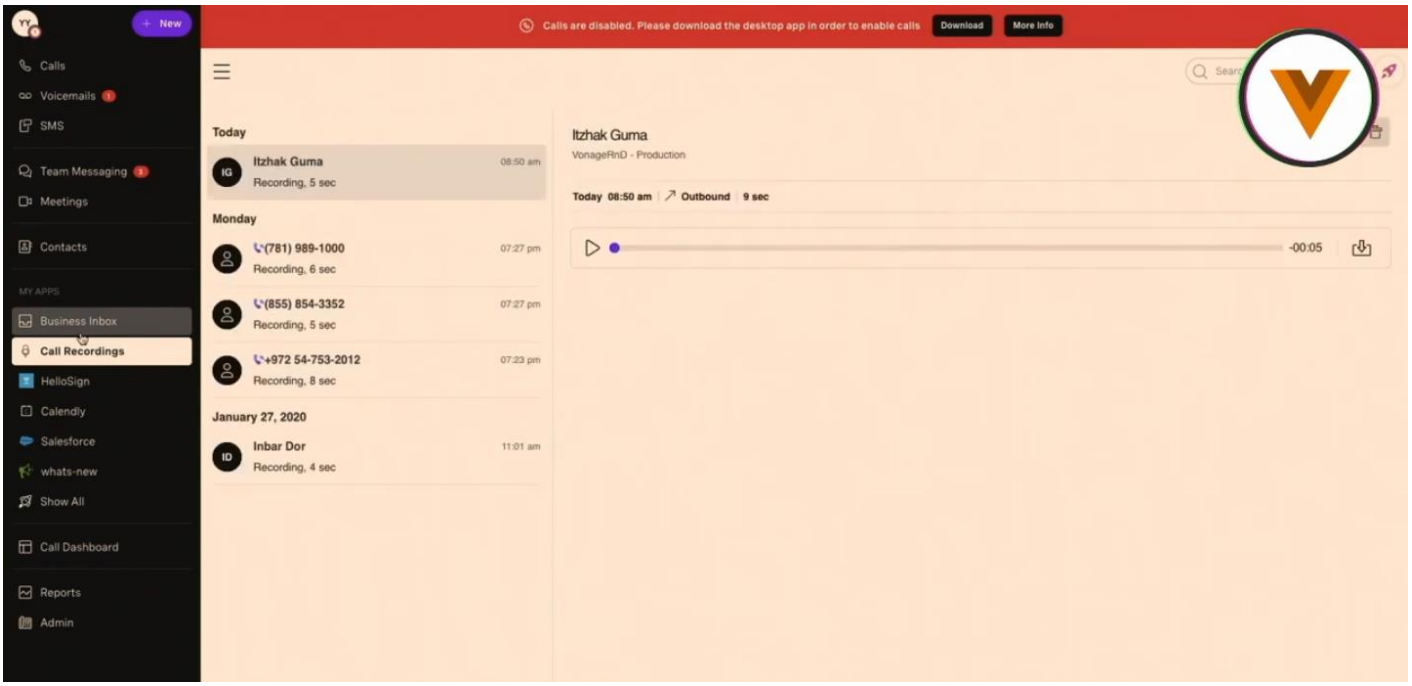

New
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
New Call
Search
Recent
Missed
Recordings
TODAY
Itzhak Guma
Outbound, 9 sec
08:50 AM
Itzhak Guma
Outbound, 17 sec
WEDNESDAY
(781) 989-1000
Missed Call
03:27 PM
Suspected Spam - CCS OFFICES
Inbar Dor
Missed Call
01:46 PM
Inbar Dor
Outbound, 8:03 min
01:46 PM
Inbar Dor
Missed Call
01:45 PM
Inbar Dor
Outbound, 0 sec
01:36 PM
TUESDAY
Inbar Dor
Outbound, 0 sec
09:41 AM
MONDAY
(781) 989-1000
Outbound, 8 sec
07:27 PM
Yoav Yanovski
(855) 854-3352
07:27 PM
Yoav Yanovski
Tomer Zilberstein
07:24 PM
Itzhak Guma
VonageRnD - Production - Extension 9175
Info
Salesforce
08:50 AM · Today
Called
To: 9175
9 sec
Contact Information
Name:
Itzhak Guma
Extension:
9175
Office:
(973) 352-0137
Company:
VonageRnD - Production
-00:05
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM

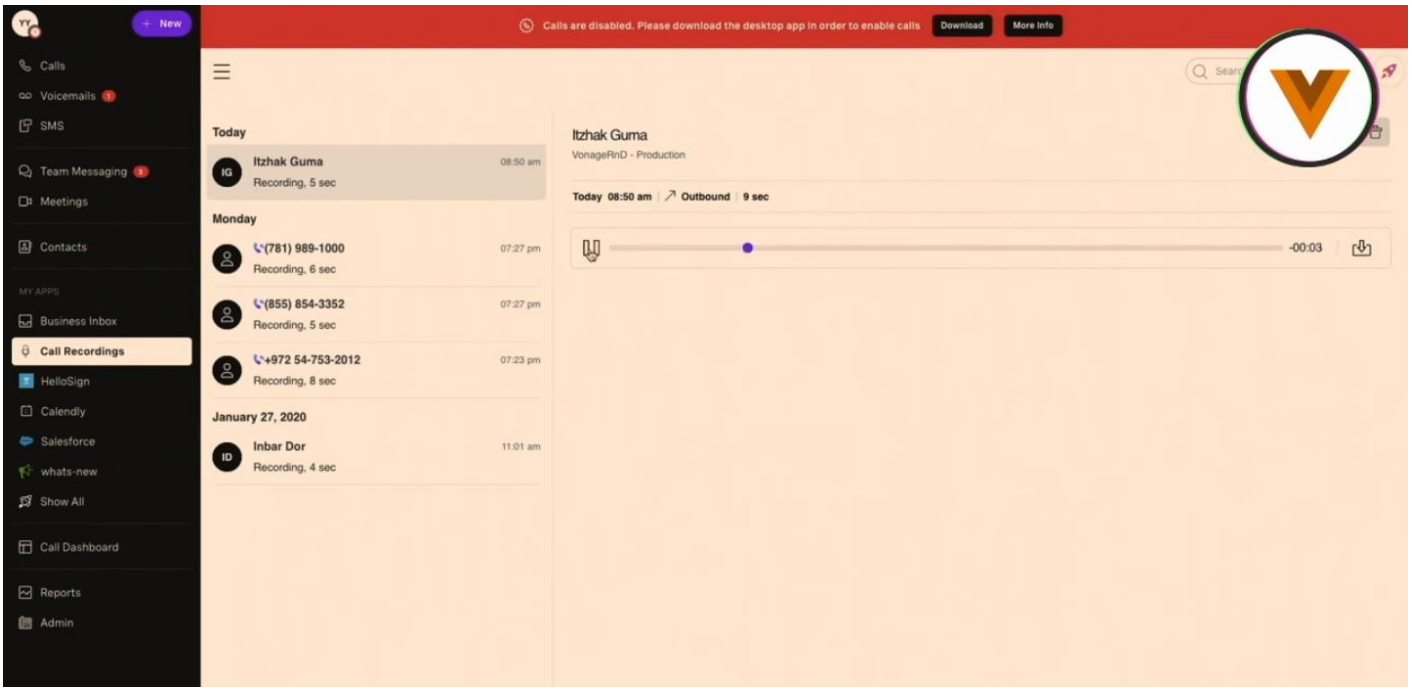

New
Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
New Call
Recent
Missed
Recordings
TODAY
Itzhak Guma
Outbound, 9 sec
08:50 AM
Itzhak Guma
Outbound, 17 sec
08:49 AM
WEDNESDAY
(781) 989-1000
Missed Call
03:27 PM
Suspected Spam - CCS OFFICES
Inbar Dor
Missed Call
01:46 PM
Inbar Dor
Outbound, 8:03 min
01:46 PM
Inbar Dor
Missed Call
01:45 PM
Inbar Dor
Outbound, 0 sec
01:36 PM
TUESDAY
Inbar Dor
Outbound, 0 sec
09:41 AM
MONDAY
(781) 989-1000
07:27 PM
Yoav Yanovski
(855) 854-3352
07:27 PM
Itzhak Guma
VonageRnD - Production - Extension 9175
Info
Salesforce
08:50 AM · Today
Called
To: 9175
9 sec
Contact Information
Name:
Itzhak Guma
Extension:
9175
Office:
(973) 352-0137
Company:
VonageRnD - Production
-00:05
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM


New
Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
New Call
Recent
Missed
Recordings
TODAY
Itzhak Guma
08:50 AM
Outbound, 9 sec
Itzhak Guma
08:49 AM
Outbound, 17 sec
WEDNESDAY
(781) 989-1000
03:27 PM
Missed Call
Suspected Spam - CCS OFFICES
Inbar Dor
01:46 PM
Missed Call
Inbar Dor
01:46 PM
Outbound, 8:03 min
Inbar Dor
01:45 PM
Missed Call
Inbar Dor
01:36 PM
Outbound, 0 sec
TUESDAY
Inbar Dor
09:41 AM
Outbound, 0 sec
MONDAY
(781) 989-1000
07:27 PM
Yoav Yanovski
(855) 854-3352
07:27 PM
Itzhak Guma
VonageRnD - Production - Extension 9175
Info
Salesforce
Select Action
Task
Add
No Activities to display
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM


New
Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Calls
New Call
Recent
Missed
Recordings
TODAY
Itzhak Guma
08:50 AM
Outbound, 9 sec
Itzhak Guma
08:49 AM
Outbound, 17 sec
WEDNESDAY
(781) 989-1000
03:27 PM
Missed Call
Suspected Spam - CCS OFFICES
Inbar Dor
01:46 PM
Missed Call
Inbar Dor
01:46 PM
Outbound, 8:03 min
Inbar Dor
01:45 PM
Missed Call
Inbar Dor
01:36 PM
Outbound, 0 sec
TUESDAY
Inbar Dor
09:41 AM
Outbound, 0 sec
MONDAY
(781) 989-1000
07:27 PM
Yoav Yanovski
(855) 854-3352
07:27 PM
Itzhak Guma
VonageRnD - Production - Extension 9175
Info
Salesforce
Task Information
Name
Scope
Related To
Subject*
CALL 9175 00:09
Comments
Status*
Assigned To* (User)
yoav yanovski
Due Date
21/02/2020
Priority*
Cancel
Create
Vue.js Amsterdam
20th & 21st February 2020
THEATER AMSTERDAM


New
Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
LOADING...


New
Calls are disabled. Please download the desktop app in order to enable calls
Download
More Info
Calls
Voicemails
SMS
Team Messaging
Meetings
Contacts
MY APPS
Business Inbox
Call Recordings
HelloSign
Calendly
Salesforce
whats-new
Show All
Call Dashboard
Reports
Admin
Today
Itzhak Guma
Recording, 5 sec
08:50 am
Monday
(781) 989-1000
Recording, 6 sec
07:27 pm
(855) 854-3352
Recording, 5 sec
07:27 pm
+972 54-753-2012
Recording, 8 sec
07:23 pm
January 27, 2020
Inbar Dor
Recording, 4 sec
11:01 am
Itzhak Guma
VonageRnD - Production
Today 08:50 am
Outbound
9 sec
-00:05



What's Next?

Where to now?

Keep in touch

- Web-IL@vonage.com
- yoav.yanovski@vonage.com
- https://www.linkedin.com/in/yoav-yanovski

Open Sources

- NPM
  https://www.npmjs.com/package/@vonage/micro-frontends
- Git
  https://github.com/Vonage/micro-frontends

We are releasing the MFC SDK and the MFE Orchestrator as OSS on GitHub and live on NPM.

@vonage/micro-frontends

0.1.0-alpha.3 • Public • Published 4 months ago

Readme | Explore BETA | 3 Dependencies | 0 Dependents | 2 Versions

@vonage/micro-frontends

About

A simple solution for Micro Frontends orchestration.

Use the MicroFrontendOrchestrator in your host application, along with the MicroFrontendComponent in the hosted components. Create your custom communication API between the host and the hosted components, and use the built in lifecycle events. Supported Micro frontend implementations are both Iframe and Web Components (native as well as Vue.js compiled web components).

Installation

```
npm install @vonage/micro-frontends
```

MicroFrontendOrchestrator (MFO)

Install

```
> npm i @vonage/micro-frontends
```

Weekly Downloads
41

Version	License
0.1.0-alpha.3	MIT
Unpacked Size	Total Files
176 kB	73
Issues	Pull Requests
0	3

Homepage
github.com/Vonage/micro-frontends#re...

MicroFrontendOrchestrator (MFO)

Use the MFO to inject and manage multiple instances of your micro frontend components.

Usage Example

Injecting a component

DOM Before Injection:

```
<div id="someDiv"></div>
```

```
import { MicroFrontendOrchestrator as MFO } from '@vonage/micro-frontends';
const myComponentDOMId = await MFO.inject('someDiv', 'myComponent', config); // m
console.log(myComponentDOMId); // prints 12345__myComponent
```

DOM After injection:

```
<div id="someDiv">
    <iframe id="12345__myComponent" src="https://example.com" />
</div>
```

API

inject(parentId: string, componentId: string, config: InjectionConfig): instanceId: string

Injects a new micro frontend application into the DOM.

Arguments

parentId - the DOM Id of the element into which the micro front end component is injected.
componentId - An Id/name that represents the injected component.
config - InjectionConfig, explained below.

github.com/Vonage/micro-frontends#re...

Repository
github.com/Vonage/micro-frontends

Last publish
4 months ago

Collaborators

Try on RunKit

Report malware

return value

instanceId - a unique id given to the specific injected component instance.

show(componentId: string): void

Shows the selected component instance. All sibling DOM nodes are automatically hidden.

Arguments

componentId - the DOM Id of the component instance you to be visible on the DOM.

remove(componentId: string)

Removes an injected component from the DOM.

Arguments

componentId - the DOM Id of the component instance to be shown.

send(componentId: string, eventId: string, eventName: string, payload: object, error: string)

Sends an event to the application.

Arguments

componentId - the DOM Id of the component instance you wish to send the event to.
eventId - a unique Id for the event, the MFO will generate it for you if you dont send any value. This argument is used to track responses sent back from hosted components.
eventName - a name that describes the type of event sent.
payload - an object passed along to the hosted component. Do not pass along functions since this object is serialized.

registerEvent(componentId, eventName, callback)

Subscribe for any event sent by the selected component instance.

Arguments

componentId - the DOM Id of the component instance from which the event is sent.
eventName - the name of the event you wish to componentId to.
callback - a function called when the event is triggered.

unsregisterEvent(componentId, eventName, callback)

Unsubscribe a previously registered event.

Arguments

componentId - the DOM Id of the component instance from which the event was sent.
eventName - the name of the event you wish to register to.
callback - a function called when the event is triggered.

getInstancesIds(componentId: string, parentId?: string): string[]

Get all instances Ids belonging to a specific componentId.

Arguments

componentId - the original componentId sent when the component was injected.
parentId - use this argument to search for all instances of a component under a specific parent element.

InjectionConfig

This is the config object used to define the injected component.

```
{
  // `type` is the type of micro frontend component you want to inject, either 'iframe' or 'webcomponent'.
  type: 'webcomponent',

  // `url` is the url used to load the remote resource, either a domain (for iframes), or a js file (for webcomponents).
  url: 'http://somedomain.com',

  // `customElementTagName` the name of the tag represents the webcomponent or custom element loaded.
  // Only needed when loading a web component.
  customElementTagName: 'my-web-component',

  // `customEvents` custom event mapping to their handler functions.
  //  you can alternatively use the registerEvent function to add custom event handlers later.
  customEvents: {
    someCustomEventName: (eventData) => { console.log (eventData) }
  },

  // `onBeforeInjected` - Function - called before the hosted component is injected into the DOM (also before the needed js code is fetched - for webcompo
  onBeforeInjected: (componentInstanceId) => { } // some custom logic before the component instance is loaded

  // `onAfterInjected` - Function - called right after the hosted component is injected into the DOM
  onAfterInjected: (componentInstanceId) => { } // some custom logic after the component instance is loaded

  // `onInitialized` - Function - called when the hosted component calls the initialize method, and is usually used to communicate component initializatio
  onInitialized: (componentInstanceId, eventId, ) => {
    // some initialization logic
    MFO.send(componentInstanceId, eventId, null, { userToken: '123', otherData: 'abc' }); // Dont forget to send back to the component it's initialization
  }

  // `onReady` - Function - called when the hosted component calls the ready method. Usually when the hosted component is ready to be displayed.
  onReady: (componentInstanceId, eventId, eventName, isReady) => {
```

```
  // `onReady` - Function - called when the hosted component calls the ready method. Usually when the hosted component is ready to be displayed.
  onReady: (componentInstanceId, eventId, eventName, isReady) => {
    // some custom logic.
    MFO.send(componentInstanceId, eventId, null, 'ACK'); // Dont forget to ack back to the component that it is now dispalyed. This resolves the promise fo
  }

  // `onError` - Function - called when the hosted component calls the ready method, or a global error occurred in the hosted component's content window(ij
  // Used to show a generic error screen instead of the hosted component, and report errors for the host application.
  onError: (componentInstanceId, eventId) => {
    // handle errors
  }

  // `onTerminate` - Function - called when the hosted component whishes to be terminated.
  onTerminate: (componentInstanceId, eventId) => {
    // Any custom logic.
    MFO.remove(componentInstanceId);
  }

  // `onBeforeRemoved` - Function - called right before the hosted component is removed from the DOM.
  onBeforeRemoved: (componentInstanceId, eventId) => { }

  // `onRemoved`- Function - called after a hosted component is removed from the DOM.
  onRemoved: (componentInstanceId, eventId) => { }
}
```

MicroFrontendComponent (MFC)

Use the MFC in your micro frontend components to report basic lifecycle events, send custom events and subscribe to host custom events.

Usage Example

```
import { MicroFrontendComponent } from '@vonage/micro-frontends';

const mfc = new MicroFrontendComponent();
const initialData = await mfc.initialize(); // get some inital data from the host app
// do some verification on your end...
verifyInitialData(initialData);
// your component is ready to be presented.
await mfc.ready();
// your component is now displayed.
```

API

All functions (except one) return a Promise. All function accept the options object with the following fields:

- requestTimeout (ms) - The promise is rejected if no answer was received from the host applicaiton after this time. Default value is 10 seconds.

More options soon...

Component Lifecycle

Component Lifecycle

initialize

Tell the host application you want to initialize a new component. This method is usually used to pass some initial verification data from the host application, such as a token.

Parameters: options Returns a Promise resolved or rejected with whatever data sent back from the host application.

```
const initialData = await mfc.initialize(); // The host application should use the onInitialized hook to respond to this event.
```

ready

Tell the host application that the component is ready to be presented. Call this function only after the call to the initialize method was successful. Parameters: options

```
await mfc.ready(options); // The host application should use the onReady hook to respond to this event.
```

error

Report an error to the host application. Parameters: options -- error - any

```
mfc.error(error, options);
```

terminate

Ask the host app to remove the component from the DOM, ending it's lifecycle. Call this function only after the call to the initialize method was successful.

Parameters: options

```
mfc.terminate(options);
```

Create a custom API between your component and the host application.

createRequest

Sends a custom request to the host application. The Promise returned by this function is resolved only when the host app sends back an answer event with the same Id as the event sent by the hosted component. The Promise is rejected when the host app sends an error, or the request times out.

```
const customData = await mfc.createRequest('myCustomEvent', { payload: { field1: '123', field2: '456'} })
```

a more common usage would be to extend the MicroFrontendComponent class and use the createRequest function to expose other functions. Example:

```
class myComponent extends MicroFrontendComponent {
    constructor() { super({}); }

    someCustomFunction(arg1, arg2, arg3) {
        // some custom logic
        return this.createRequest('someCustomEvent', {payload: { a: arg1, b: arg2, c: arg3 }, requestTimeout: 500})
    }
}
```

To resolve or reject the promise, the host applicaiton should:

1. use the customEvents option to map the event sent by the hosted component to a callback function.
2. The host callback function should eventually use the mfc.send(eventId, eventName, payload) function to respond to the request.

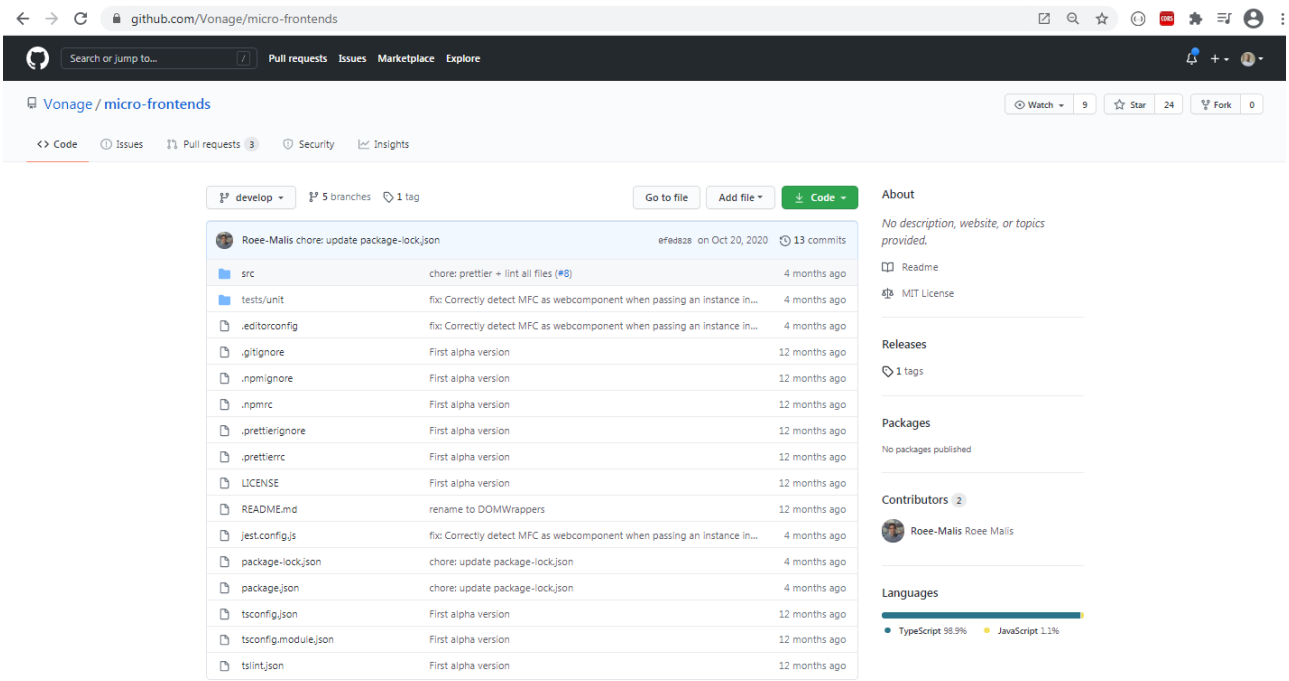
registerEvent

This is the only function that does not return a Promise. Use this function to register to events that were initiated by the host application. The callback is called with the payload sent by the host application.

Parameters:

- eventName - the event to be registered to.
- callBack - function to be executed when the event is sent.

```
mfc.registerEvent('someCustomEvent', (eventData) => { console.log(eventData) });
```



README.md

@vonage/micro-frontends

About

A simple solution for Micro Frontends orchestration.

Use the MicroFrontendOrchestrator in your host application, along with the MicroFrontendComponent in the hosted components. Create your custom communication API between the host and the hosted components, and use the built in lifecycle events. Supported Micro frontend implementations are both Iframe and Web Components (native as well as Vue.js compiled web components).

Installation

```
npm install @vonage/micro-frontends
```

MicroFrontendOrchestrator (MFO)

Use the MFO to inject and manage multiple instances of your micro frontend components.

Usage Example

Injecting a component

DOM Before Injection:

```
<div id="someDiv"></div>
```

```
import { MicroFrontendOrchestrator as MFO } from '@vonage/micro-frontends';
const myComponentDOMId = await MFO.inject('someDiv', 'myComponent', config); // more info on the config object
console.log(myComponentDOMId); // prints 12345__myComponent
```

DOM After injection:

```
<div id="someDiv">
    <iframe id="12345__myComponent" src="https://example.com" />
</div>
```

API

inject(parentId: string, componentId: string, config: InjectionConfig): instanceId: string

Injects a new micro frontend application into the DOM.

Arguments

parentId - the DOM Id of the element into which the micro front end component is injected.
componentId - An Id/name that represents the injected component.
config - InjectionConfig, explained below.

return value

instanceId - a unique id given to the specific injected component instance.

show(componentId: string): void

Shows the selected component instance. All sibling DOM nodes are automatically hidden.

Arguments

componentId - the DOM Id of the component instance you to be visible on the DOM.

remove(componentId: string)

Removes an injected component from the DOM.

Arguments

componentId - the DOM Id of the component instance to be shown.

send(componentId: string, eventId: string, eventName: string, payload: object, error: string)

Sends an event to the application.

Arguments

componentId - the DOM Id of the component instance you wish to send the event to.
eventId - a unique Id for the event, the MFO will generate it for you if you dont send any value. This argument is used to track responses sent back from hosted components.
eventName - a name that describes the type of event sent.
payload - an object passed along to the hosted component. Do not pass along functions since this object is serialized.

registerEvent(componentId, eventName, callback)

Subscribe for any event sent by the selected component instance.

Arguments

componentId - the DOM Id of the component instance from which the event is sent.
eventName - the name of the event you wish to componentId to.
callback - a function called when the event is triggered.

unsregisterEvent(componentId, eventName, callback)

Unsubscribe a previously registered event.

Arguments

componentId - the DOM Id of the component instance from which the event was sent.
eventName - the name of the event you wish to register to.
callback - a function called when the event is triggered.

getInstancesIds(componentId: string, parentId?: string): string[]

Get all instances Ids belonging to a specific componentId.

Arguments

componentId - the original componentId sent when the component was injected.
parentId - use this argument to search for all instances of a component under a specific parent element.

InjectionConfig

This is the config object used to define the injected component.

```
{
  // `type` is the type of micro frontend component you want to inject, either 'iframe' or 'webcomponent'.
  type: 'webcomponent',

  // `url` is the url used to load the remote resource, either a domain (for iframes), or a js file (for
webcomponents).
  url: 'http://somedomain.com',

  // `customElementTagName` the name of the tag represents the webcomponent or custom element loaded.
  // Only needed when loading a web component.
  customElementTagName: 'my-web-component',

  // `customEvents` custom event mapping to their handler functions.
  //  you can alternatively use the registerEvent function to add custom event handlers later.
  customEvents: {
    someCustomEventName: (eventData) => { console.log (eventData) }
  },

  // `onBeforeInjected` - Function - called before the hosted component is injected into the DOM (also
before the needed js code is fetched - for webcomponents).
  onBeforeInjected: (componentInstanceId) => { } // some custom logic before the component instance is
loaded

  // `onAfterInjected` - Function - called right after the hosted component is injected into the DOM
  onAfterInjected: (componentInstanceId) => { } // some custom logic after the component instance is loaded

  // `onInitialized` - Function - called when the hosted component calls the initialize method, and is
usually used to communicate component initialization and token exchange to the hosted component.
  onInitialized: (componentInstanceId, eventId, ) => {
    // some initialization logic
    MFO.send(componentInstanceId, eventId, null, { userToken: '123', otherData: 'abc' }); // Dont forget to
send back to the component it's initialization data. This resolves the promise for the 'initialize' method
in the hosted component.
  }

  // `onReady` - Function - called when the hosted component calls the ready method. Usually when the
hosted component is ready to be displayed.
  onReady: (componentInstanceId, eventId, eventName, isReady) => {
    // some custom logic.
    MFO.send(componentInstanceId, eventId, null, 'ACK'); // Dont forget to ack back to the component that
it is now dispalyed. This resolves the promise for the 'ready' method in the hosted component.
  }

  // `onError` - Function - called when the hosted component calls the ready method, or a global error
occurred in the hosted component's content window(iframe only).
  // Used to show a generic error screen instead of the hosted component, and report errors for the host
application.
  onError: (componentInstanceId, eventId) => {
```

```
occurred in the hosted component's content window(iframe only).
  // Used to show a generic error screen instead of the hosted component, and report errors for the host
application.
  onError: (componentInstanceId, eventId) => {
    // handle errors
  }

  // `onTerminate` - Function - called when the hosted component whishes to be terminated.
  onTerminate: (componentInstanceId, eventId) => {
    // Any custom logic.
    MFO.remove(componentInstanceId);
  }

  // `onBeforeRemoved` - Function - called right before the hosted component is removed from the DOM.
  onBeforeRemoved: (componentInstanceId, eventId) => { }

  // `onRemoved`- Function - called after a hosted component is removed from the DOM.
  onRemoved: (componentInstanceId, eventId) => { }
}
```

MicroFrontendComponent (MFC)

Use the MFC in your micro frontend components to report basic lifecycle events, send custom events and subscribe to host custom events.

Usage Example

```
import { MicroFrontendComponent } from '@vonage/micro-frontends';

const mfc = new MicroFrontendComponent();
const initialData = await mfc.initialize(); // get some inital data from the host app
// do some verification on your end...
verifyInitialData(initialData);
// your component is ready to be presented.
await mfc.ready();
// your component is now displayed.
```

API

All functions (except one) return a Promise. All function accept the options object with the following fields:

- requestTimeout (ms) - The promise is rejected if no answer was received from the host applicaiton after this time. Default value is 10 seconds.

More options soon...

Component Lifecycle

initialize

Tell the host application you want to initialize a new component. This method is usually used to pass some initial verification data from the host application, such as a token.

Parameters: options Returns a Promise resolved or rejected with whatever data sent back from the host application.

```
const initialData = await mfc.initialize(); // The host application should use the onInitialized hook to
respond to this event.
```

ready

Tell the host application that the component is ready to be presented. Call this function only after the call to the initialize method was successful. Parameters: options

```
await mfc.ready(options); // The host application should use the onReady hook to respond to this event.
```

error

Report an error to the host application. Parameters: options -- error - any

```
mfc.error(error, options);
```

terminate

Ask the host app to remove the component from the DOM, ending it's lifecycle. Call this function only after the call to the initialize method was successful.

Parameters: options

```
mfc.terminate(options);
```

Create a custom API between your component and the host application.

createRequest

Sends a custom request to the host application. The Promise returned by this function is resolved only when the host app sends back an answer event with the same Id as the event sent by the hosted component. The Promise is rejected when the host app sends an error, or the request times out.

```
const customData = await mfc.createRequest('myCustomEvent', { payload: { field1: '123', field2: '456'} })
```

a more common usage would be to extend the MicroFrontendComponent class and use the createRequest function to expose other functions. Example:

```
class myComponent extends MicroFrontendComponent {
    constructor() { super({}); }

    someCustomFunction(arg1, arg2, arg3) {
        // some custom logic
        return this.createRequest('someCustomEvent', {payload: { a: arg1, b: arg2, c: arg3 },
requestTimeout: 500})
    }
}
```

To resolve or reject the promise, the host applicaiton should:

1. use the customEvents option to map the event sent by the hosted component to a callback function.
2. The host callback function should eventually use the mfc.send(eventId, eventName, payload) function to respond to the request.

registerEvent

This is the only function that does not return a Promise. Use this function to register to events that were initiated by the host application. The callback is called with the payload sent by the host application.

Parameters:

- eventName - the event to be registered to.
- callBack - function to be executed when the event is sent.

```
mfc.registerEvent('someCustomEvent', (eventData) => { console.log(eventData) });
```