
+



Speaker

Meet the team



Stefan Benicke

Senior Frontend Developer

JavaScript and CSS enthusiast with more than 15 years of experience.



Thomas Stenitzer

Technical Marketing Specialist

Problem solver & solution finder, HTML since '97

eyeson video meeting

Short introduction

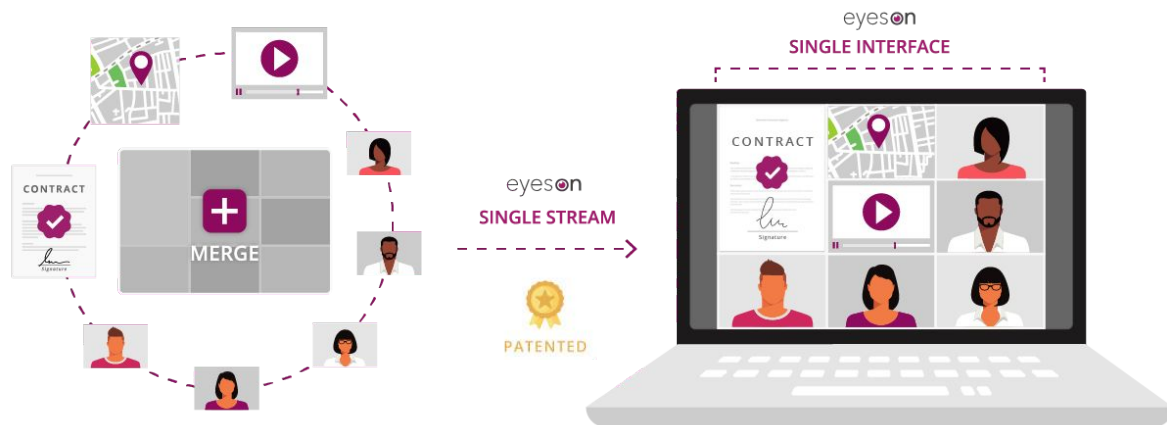
MCU based SingleStream - everyone sees the same.

wysiwyg - also on recording/snapshot and broadcast

Programmable via API (layout, layer, playback, and more)

Insert live data into the video

Use free default UI or create your own



eyeson video meeting

What makes it so special?

- eyeson integration is easy as 1,2,3
- Provides default UI with many options like "show_label", "exit_url", "logo", "hide_chat" etc.
- Provides webhooks for state changes for meeting, recordings, participants

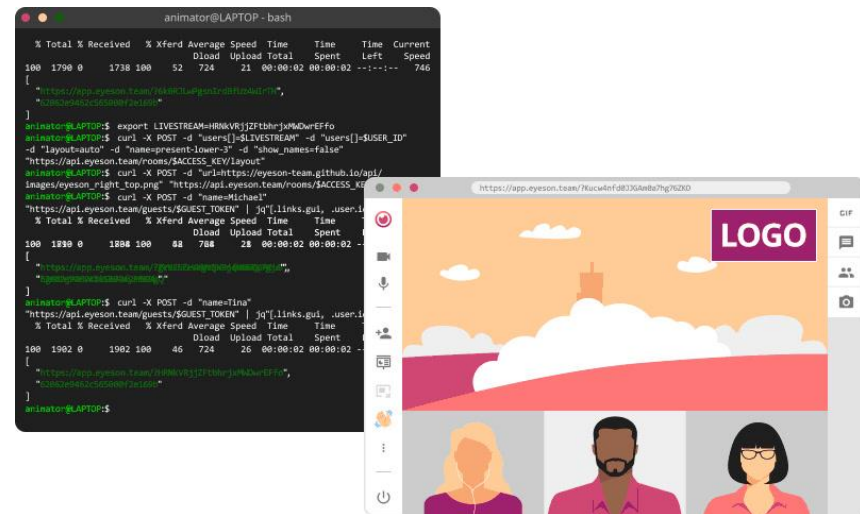
eyeson video meeting

Short introduction

eyeson API offers many features to control the meeting and the video stream.


- Arrange users on the podium with **layout** API
- Add background and/or foreground images with **layer** API
- Inject media with **playback** API

Combine all 3 to create virtual rooms like cinema, newsroom, consulting room



Get your hands on!

What you need

- Your **API key**
sign up at developers.eyeson.team
or simply visit our booth
@ Hall A / Level 1 / Booth A3.7
- Github repository
 [/eyeson-team/node-api-example-app](https://github.com/eyeson-team/node-api-example-app)



Demo setup

Navigate to your desired chapter folder
(for example "06-final-version").

Add your API key to the ".env" config file.

```
$ cd 06-final-version
$ cp .env.example .env
$ vim .env // enter API key
$ npm install
$ npm start
```

API key vs. ACCESS key

When and why?

There are two levels of authorization: team-based via API key and user-based temporary ACCESS key.

API key

- Start/stop meeting
- Add participants to the meeting
- Register webhook
- Get recording info

ACCESS key

Any other communication with the API requires user-based authorization. An active room and user is identified by an ACCESS key.



I. start meeting

```
curl -X POST \  
-H "Authorization: YOUR_API_KEY" \  
-d "user[name]=John Doe" \  
https://api.eyeson.team/rooms
```

```
import eyesonAPI from 'eyeson-node'  
  
const eyeson = new eyesonAPI({ apiKey: process.env.API_KEY })  
  
const client = await eyeson.join('John Doe')
```

That's easy - isn't it?

Username is required.

Meeting should only be started when needed, so this user should join immediately!

I. start meeting

Many options available

```
curl -X POST \  
-H "Authorization: YOUR_API_KEY" \  
-d "id=daily-standup" \  
-d "name=Daily Standup" \  
-d "user[id]=john.doe@mycompany.com" \  
-d "user[name]=John Doe" \  
-d "options[exit_url]=https://mycompany.com/meeting-exit" \  
-d "options[broadcast_available]=false" \  
-d "options[lock_available]=true" \  
-d "options[custom_fields][logo]=https://mycompany.com/images/icon.png" \  
-d "options[custom_fields][virtual_background]=true" \  
-d "options[custom_fields][virtual_background_allow_guest]=true" \  
https://api.eyeson.team/rooms
```

Options for default UI can be included!

I. start meeting

Many options available

```
import eyesonAPI from 'eyeson-node'

const eyeson = new eyesonAPI({ apiKey: process.env.API_KEY })

const client = await eyeson.join('John Doe', 'daily-standup', {
  name: 'Daily Standup',
  user: {
    id: 'john.doe@mycompany.com'
  },
  options: {
    exit_url: 'https://mycompany.com/meeting-exit',
    broadcast_available: false,
    lock_available: true,
    custom_fields: {
      logo: 'https://mycompany.com/images/icon.png',
      virtual_background: true,
      virtual_background_allow_guest: true
    }
  }
})
// client.links.gui
```

I. start meeting

Join another user

```
curl -X POST \  
-H "Authorization: YOUR_API_KEY" \  
-d "id=daily-standup" \  
-d "user[id]=bart.simpson@mycompany.com" \  
-d "user[name]=Bart Simpson" \  
https://api.eyeson.team/rooms
```

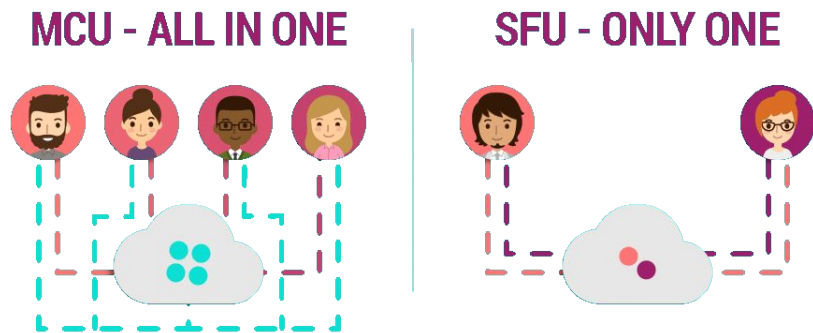
```
const client2 = await eyeson.join('Bart Simpson', 'daily-standup', {  
  user: {  
    id: 'bart.simpson@mycompany.com'  
  }  
})  
// client2.links.gui
```

Quick Tip: use “guest-links” to allow quick join of unregistered users

SFU vs. MCU mode

- SFU forwards the client's a/v stream to the other client
- In snapshot/recording/broadcast still all-in-one MCU is used
- SFU only visible in UI
- In SFU mode the remote stream does NOT contain any layer or layout

=> sfu_mode = "disabled"



```
curl -X POST \  
-H "Authorization: YOUR_API_KEY" \  
-d "user[name]=Bart Simpson" \  
-d "options[sfu_mode]=disabled" \  
https://api.eyeson.team/rooms
```

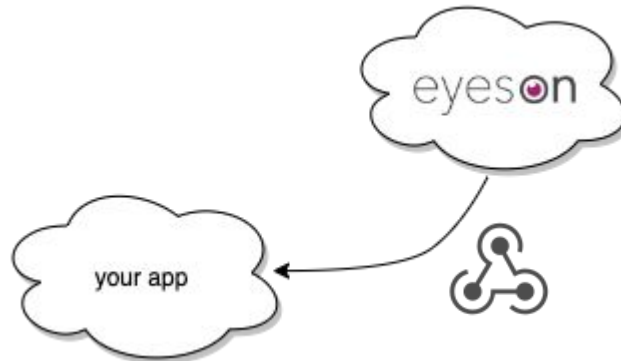
```
const client = await eyeson.join('Bart Simpson', null, {  
  options: {  
    'sfu_mode': 'disabled'  
  }  
})
```

II. webhook

Why webhook?

Get notified when:

- State change information for meeting/participant
- Snapshot/recording is created
- Automate tasks



II. webhook

Register your app

```
curl -X POST \  
-H "Authorization: YOUR_API_KEY" \  
-d "url=https://mycompany.com/webhook" \  
-d "types=room_update,recording_update" \  
https://api.eyeson.team/webhooks
```

=> Webhooks are related to all meetings that started with same API key!

III. layout

Arrange video participants on the video stream, even overlapping is possible. predefined set of named layouts available. if you need more -> tell us

```
curl -X POST \  
-d "layout=auto" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layout
```

```
client.setLayout({ layout: 'auto' })
```

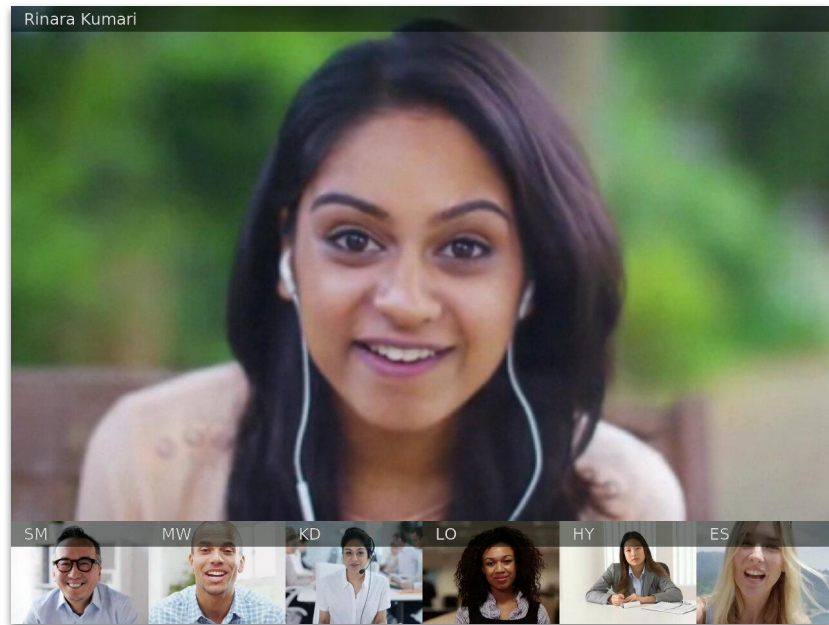
“auto” layout grows with number of participants - 1, 2, 4, 6, 9

III. layout

Example: large speaker,
6 autofilled places underneath

```
curl -X POST \  
-d "layout=auto" \  
-d "name=present-upper-6" \  
-d "users[]=bart.simpson@mycompany.com" \  
-d "voice_activation=true" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layout
```

```
client.setLayout({  
  layout: 'auto',  
  name: 'present-upper-6',  
  users: ['bart.simpson@mycompany.com'],  
  voice_activation: true  
})
```



“voice activation” ... hidden users will be shown as soon as they start talking

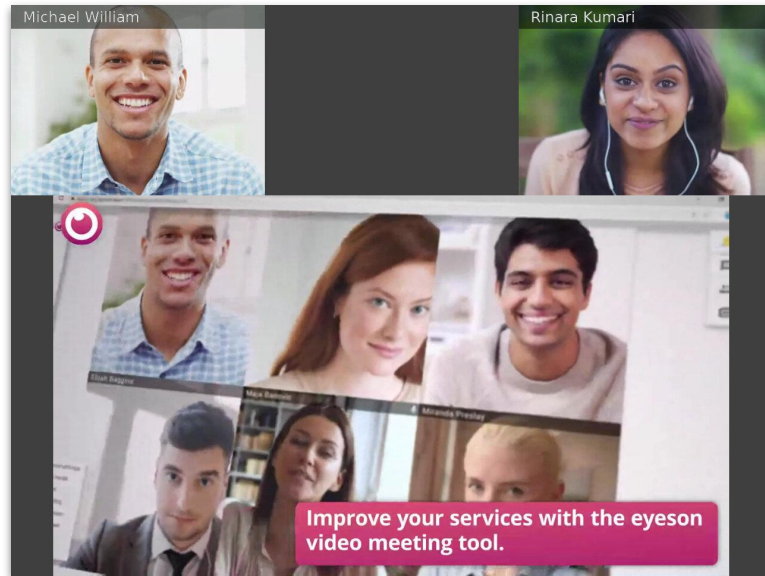
III. layout

Example: 2 fix speakers with large presentation space

```
curl -X POST \
-d "layout=custom" \
-d "name=present-lower-3" \
-d "users[]=playback" \
-d "users[]=john.doe@mycompany.com" \
-d "users[]" \
-d "users[]=bart.simpson@mycompany.com" \
https://api.eyeson.team/rooms/ACCESS_KEY/layout
```

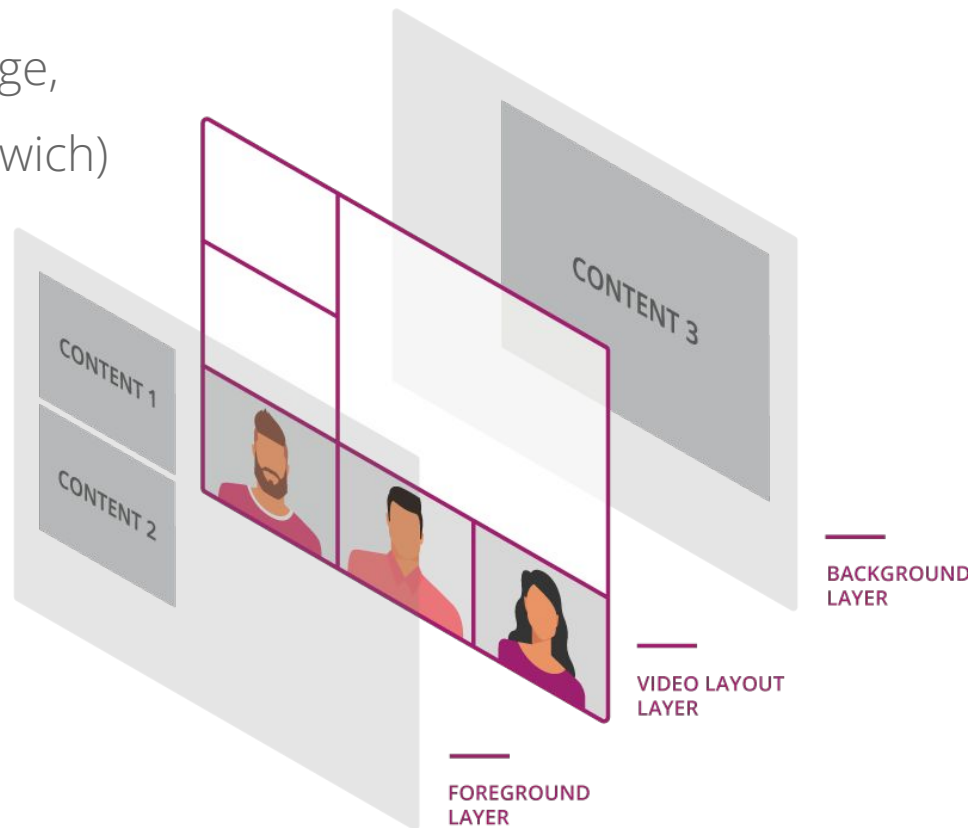
```
client.setLayout({
  layout: 'custom',
  name: 'present-lower-3',
  users: [
    'playback',
    'john.doe@mycompany.com',
    '',
    'bart.simpson@mycompany.com'
  ]
})
```

* note the empty place will stay empty



IV. layer

Background and/or foreground image,
video participants in-between (sandwich)



IV. layer

Text to overlay

```
curl -X POST \
-d "insert[icon]=https://mycompany.com/images/icon.png" \
-d "insert[content]=text message in the video stream" \
https://api.eyeson.team/rooms/ACCESS_KEY/layers
```

```
client.setLayer({
  insert: {
    icon: 'https://mycompany.com/images/icon.png',
    content: 'text message in the video stream'
  }
})
```



IV. layer

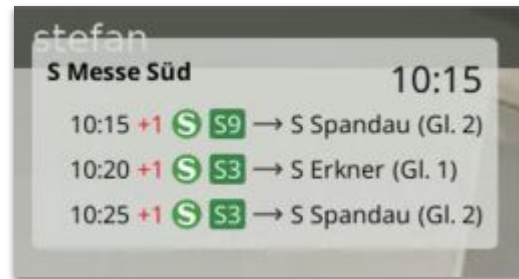
Example: company logo as static overlay

```
curl -X POST \  
-d "url=https://mycompany.com/images/logo-overlay.png" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layers
```

```
client.setLayer({ url: 'https://mycompany.com/images/logo-overlay.png' })
```

IV. layer

Example: live data from public transport



- Grab and process data
- Draw data on canvas (1280 x 960 px) on desired position (rest is transparent)
- Send to layer API

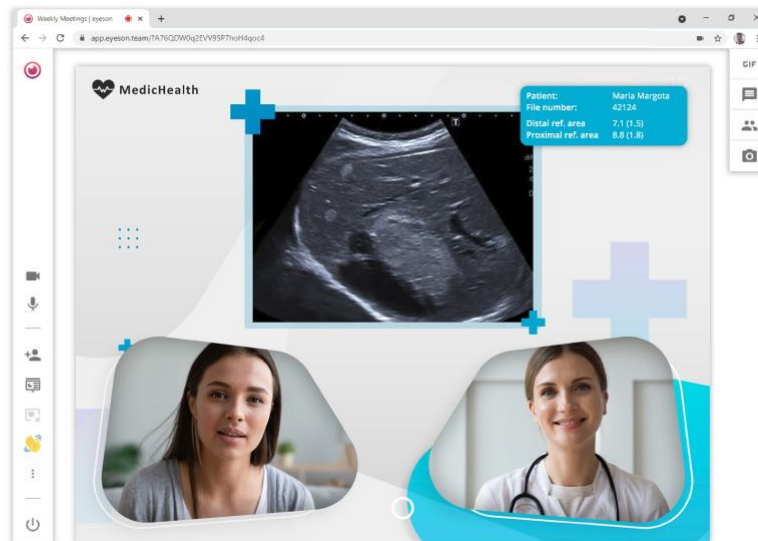
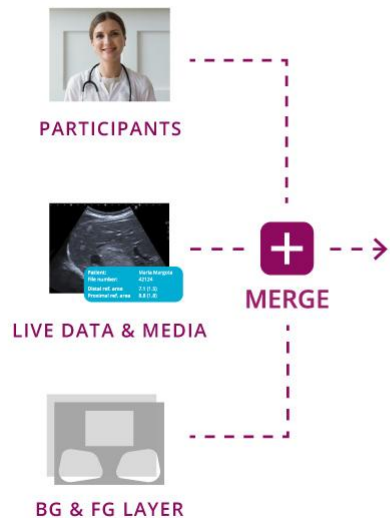
```
const data = await getPublicTransport()
const file = createOverlayFromCanvas(data)
const formData = new FormData()
formData.append('file', file)
await fetch(`https://api.eyeson.team/rooms/${accessKey}/layers`, { method: 'POST', body: formData })
```

(function names only for demonstration purpose)

V. virtual rooms

Combine layout and layer to create virtual rooms

Everyone sees the same, even on recording/snapshot and broadcasting! => **SingleStream**



V. virtual rooms

Example: consulting room

```
curl -X POST \  
-d "layout=auto" \  
-d "name=present-upper-3-bg" \  
-d "users[]=playback" \  
-d "users[]=john.doe@mycompany.com" \  
-d "users[]=placeholder" \  
-d "users[]" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layout
```

- Note the last user position is empty and since layout is "auto", your guest will automatically displayed here.
- "placeholder" is already occupied.

```
curl -X POST \  
-d "url=https://mycompany.com/images/consulting-room-foreground.png" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layers
```

```
curl -X POST \  
-d "url=https://mycompany.com/images/consulting-room-background.png" \  
-d "z-index=-1" \  
https://api.eyeson.team/rooms/ACCESS_KEY/layers
```

V. virtual rooms

Example: consulting room

```
client.setLayout({
  layout: 'auto',
  name: 'present-upper-3-bg',
  users: [
    'autoplay',
    'john.doe@mycompany.com',
    'placeholder',
  ]
})
```

```
client.setLayer({ url: 'https://mycompany.com/images/consulting-room-foreground.png' })
```

```
client.setLayer({
  url: 'https://mycompany.com/images/consulting-room-background.png',
  'z-index': '-1'
})
```

- Note the last user position is empty and since layout is "auto", your guest will automatically displayed here.
- "placeholder" is already occupied.

V. virtual rooms

Add live data

```
const data = getCurrentData()
const canvas = createCanvasWithForegroundImage('https://mycompany.com/images/consulting-room-foreground.png')
const file = addDataToCanvas(canvas, data)
const formData = new FormData()
formData.append('file', file)
await fetch(`https://api.eyeson.team/rooms/${accessKey}/layers`, { method: 'POST', body: formData })
```

(function names only for demonstration purpose)

Add live data to canvas and update on changes

VI. playback

```
curl -X POST \  
-d "playback[audio]=true" \  
-d "playback[play_id]=playback" \  
-d "playback[url]=https://mycompany.com/media/instruction.webm" \  
https://api.eyeson.team/rooms/ACCESS_KEY/playbacks
```

play **MP4** or **WEBM** video file
from any public URL
(WEBM preferred)

```
client.startPlayback({  
  playback: {  
    audio: true,  
    play_id: 'playback',  
    url: 'https://mycompany.com/media/instruction.webm'  
  }  
})
```

- use "*play_id*" in combination with named user place in layout
- additional "*replacement_id*" to replace user for the time of the playback

VII. custom mediastream



Instead of user's webcam, you can use our tools to stream more content like IP-Cams, Drones, Web-content like games, live-streams and more.



Ghost - "Go Host Streaming Client"

Web-SDK - external MediaStream feature



VIII. automation

Possibilities / examples

- Store recordings/snapshots immediately (eg. to archive)
- Adjust layout when certain user joins/leaves
- Set company logo on meeting start
- Post snapshot immediately
- Log meeting state



Get Connected!

Start exploring

Register your own free API key

[SIGN UP](#)

Want a demo?

Schedule your personal live demo.

[SCHEDULE NOW](#)

Join dev community?

Apply via partner application form.

[APPLY NOW](#)

Join our team :)

One of us! One of us!
One of us!

[APPLY NOW](#)

