

FIRAT ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
BMÜ401 – BİLGİSAYAR
MÜHENDİSLİĞİ.TASARIM



PROJE ADI: Görüntülü mesajlaşma uygulaması

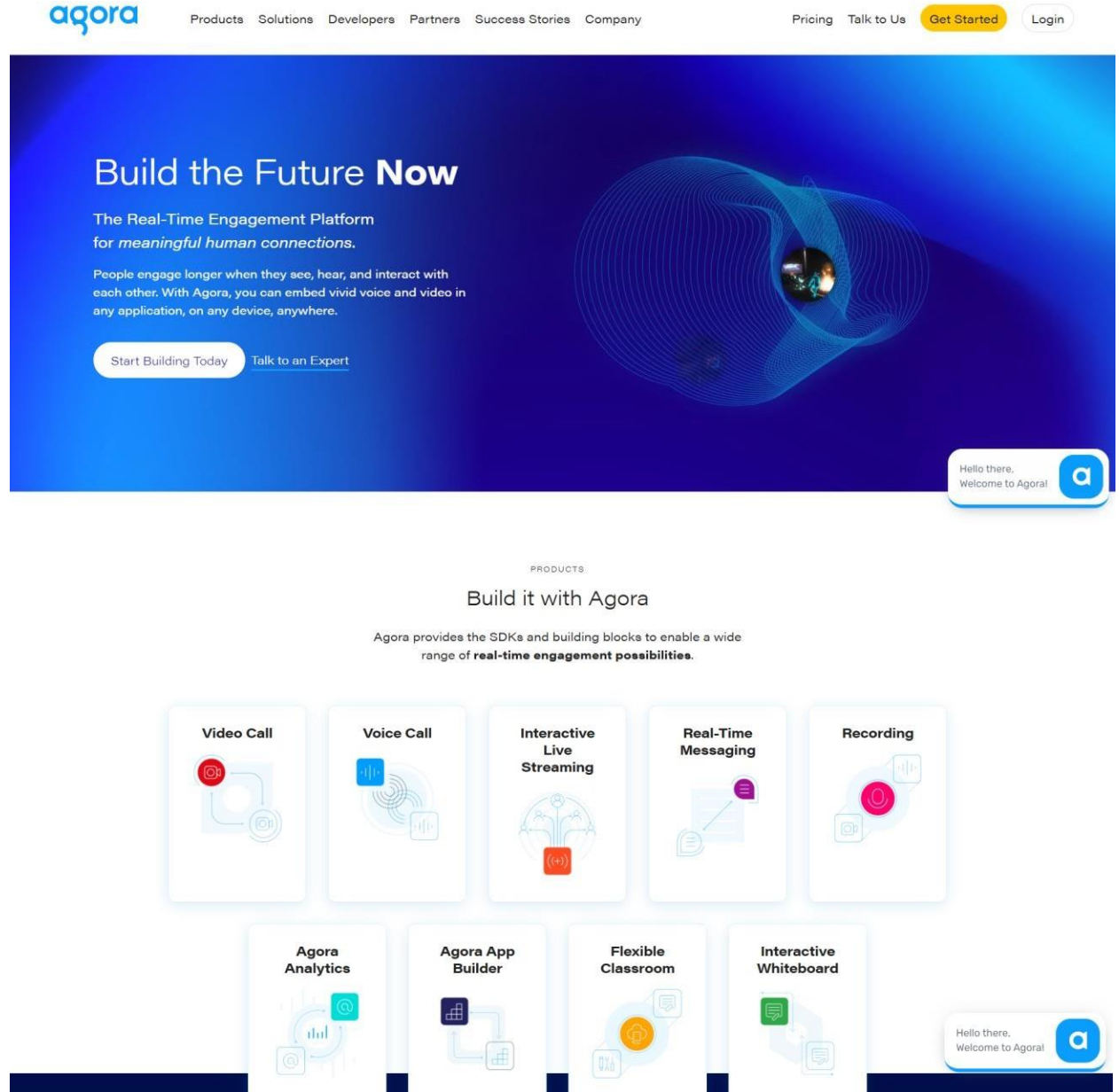
PROJE EKİBİ:

- 180260611-ABDULKADİR SULTANİELEŞREFİ
- 180260601-AYAH HASAN

Projenin Raporu

Projede kullanılan teknoloji agora teknoloji.

Agora nedir: Anlamlı insan bağlantıları için Gerçek Zamanlı Katılım Platformu



The image shows the Agora website landing page. The header features the Agora logo on the left, a navigation menu with links to Products, Solutions, Developers, Partners, Success Stories, and Company in the center, and a right-hand section with links for Pricing, Talk to Us, a yellow Get Started button, and a Login button. The main hero section has a dark blue background with a glowing blue sphere containing a globe. The text 'Build the Future Now' is prominently displayed, followed by the tagline 'The Real-Time Engagement Platform for meaningful human connections.' Below this, a paragraph states: 'People engage longer when they see, hear, and interact with each other. With Agora, you can embed vivid voice and video in any application, on any device, anywhere.' Two buttons are present: 'Start Building Today' and 'Talk to an Expert'. A small chat bubble in the bottom right corner says 'Hello there, Welcome to Agora!' with the Agora logo. The 'PRODUCTS' section is titled 'Build it with Agora' and includes the text: 'Agora provides the SDKs and building blocks to enable a wide range of real-time engagement possibilities.' Below this, there are nine product cards arranged in two rows. The top row contains: Video Call, Voice Call, Interactive Live Streaming, Real-Time Messaging, and Recording. The bottom row contains: Agora Analytics, Agora App Builder, Flexible Classroom, and Interactive Whiteboard. Each card features an icon representing its function. A second chat bubble is located at the bottom right of the products section.

agora

Products Solutions Developers Partners Success Stories Company

Pricing Talk to Us Get Started Login

Build the Future Now

The Real-Time Engagement Platform for meaningful human connections.

People engage longer when they see, hear, and interact with each other. With Agora, you can embed vivid voice and video in any application, on any device, anywhere.

Start Building Today Talk to an Expert

Hello there, Welcome to Agora!

PRODUCTS

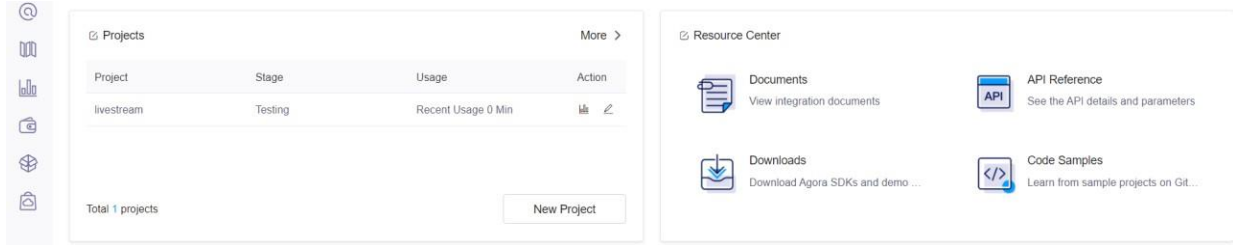
Build it with Agora

Agora provides the SDKs and building blocks to enable a wide range of real-time engagement possibilities.

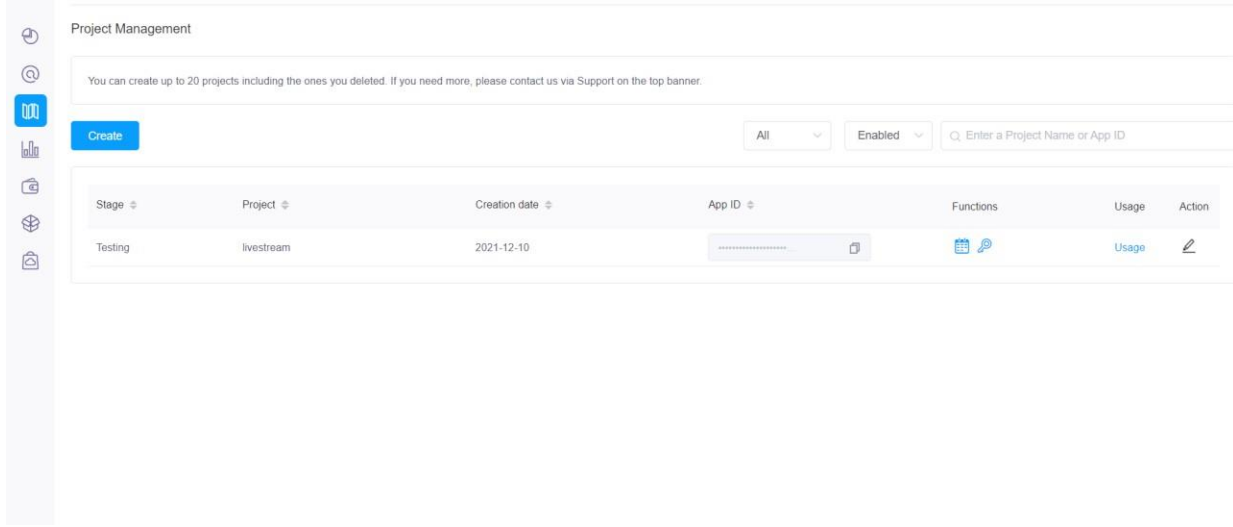
- Video Call
- Voice Call
- Interactive Live Streaming
- Real-Time Messaging
- Recording
- Agora Analytics
- Agora App Builder
- Flexible Classroom
- Interactive Whiteboard

Hello there, Welcome to Agora!

Projeyi alıřtırmak iin agora.io web sitesinde hesap amak gerekiyor.
Hesap atıktan sonra agorada projeyi yapmak.



Projeyi oluřturduktan sonra sol taraftaki project menegment butonuna bastıėımızda ařaėıdaki grsel ıkacak.



Sonra anahtar simgesine basıp Temp Token oluřturmamız gerekiyor.

Token

Temp Token for audio/video call does not support RTM. For more information, please view documentation [here](#)

APP ID: [redacted]

App certificate: [redacted]

Channel Name:

Temp Token: Generate Temp Token Back

[To learn how to deploy a token server, please click here](#)

Önce istediğimiz Channal Name ismi giriliyor sonra Generate Temp Token butonuna basıyoruz.

APP ID , Channel Name ve Token projeye eklenmelidir.

Aşağıdaki görselde nerede eklencekleri gösteriliyor.

```
> assets
> index.html
ngrok.exe
script.js
style.css

1  // #1
2  let client = AgoraRTC.createClient({mode: 'rtc', codec: 'vp8'})
3
4  // #2
5  let config = {
6    appid: null,
7    token: null,
8    uid: null,
9    channel: null,
10  }
11
```

Kod kısmı:

```
let client = AgoraRTC.createClient({mode: 'rtc', codec: 'vp8'})
```

kullanıcı oluşturmak için kullanılan ilk komut

Agora API Reference

Agora Core Methods

createClient

• createClient(config: *ClientConfig*): *IAgoraRTCClient*

Creates a local client object for managing a call.

This is usually the first step of using the Agora Web SDK.

Parameters

▪ config: *ClientConfig*

The configurations for the client object, including channel profile and codec. The default codec is `vp8` and default channel profile is `rtc`. See *ClientConfig* for details.

Returns *IAgoraRTCClient*

```
// #3 - Setting tracks for when user joins
let localTracks = {
  audioTrack:null,
  videoTrack:null
}

// #4 - Want to hold state for users audio and video so user can mute and hide
let localTrackState = {
  audioTrackMuted:false,
  videoTrackMuted:false
}

// #5 - Set remote tracks to store other users
let remoteTracks = {}
```

daha sonra kullanılacak nesneler

```
document.getElementById('join-btn').addEventListener('click', async () => {  
  config.uid = document.getElementById('username').value  
  await joinStreams()  
  document.getElementById('join-wrapper').style.display = 'none'  
  document.getElementById('footer').style.display = 'flex'  
})
```

Join butonuna olay(event) ekleniyor

Join butonuna basıldığı zaman bu kodlar gerçekleştiriliyor

```
config.uid = document.getElementById('username').value
```

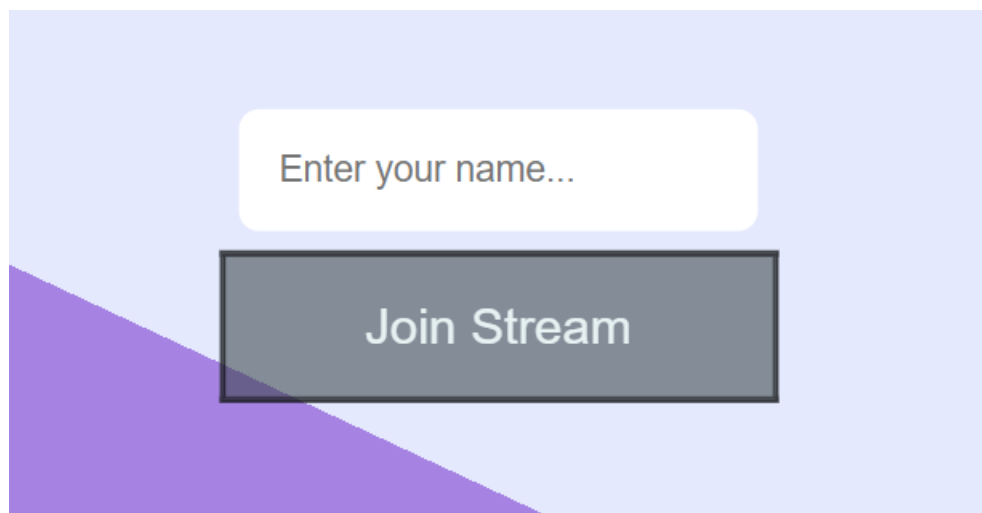
Html'den veri almak için

```
await joinStreams()
```

await kullanmak için fonksiyondan önce async eklemek gerekiyor

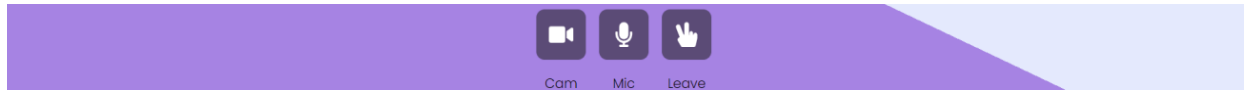
await işi ise joinStreams() fonksiyonu bitene kadar bekler bittiği zaman sonraki satıra geçer

```
document.getElementById('join-wrapper').style.display = 'none'
```



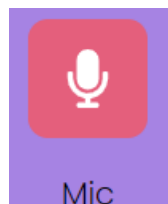
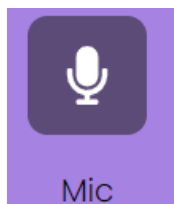
Join Stream butonu ve inputu gizler

```
document.getElementById('footer').style.display = 'flex'
```



Yukardaki butonlar gösterir.

```
document.getElementById('mic-btn').addEventListener('click', async () => {  
  //Check if what the state of muted currently is  
  //Disable button  
  if(!localTrackState.audioTrackMuted){  
    //Mute your audio  
    await localTracks.audioTrack.setMuted(true);  
    localTrackState.audioTrackMuted = true  
    document.getElementById('mic-btn').style.backgroundColor = 'rgb(255, 80,  
80, 0.7)'  
  }else{  
    await localTracks.audioTrack.setMuted(false)  
    localTrackState.audioTrackMuted = false  
    document.getElementById('mic-btn').style.backgroundColor = '#1f1f1f8e'  
  }  
})
```



Mikrofon butonun olayı(event)

```
if(!localTrackState.audioTrackMuted){
```

daha önce tanımladığımız localTrackState nesnesi içindeki
audioTrackMuted false ile başlatıldı

eğer ses kapalı değilse gir

```
await localTracks.audioTrack.setMuted(true);
```

localTracks:daha önce tanımladığımız nesne

audioTrack:join butonuna basıldıktan sonra audioTrack a agoradan bir
metod atacağız

```
audioTrack =AgoraRTC.createMicrophoneAudioTrack()
```

setMuted: agorada kullanılan bir metod

```
localTrackState.audioTrackMuted = true
```

ses kapalı =true

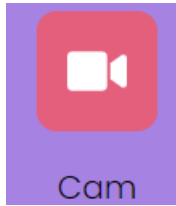
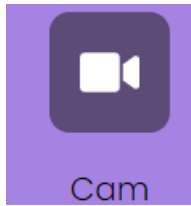
```
document.getElementById('mic-btn').style.backgroundColor ='rgb(255, 80, 80, 0.7)'
```

mic butonunun arkasındaki rengi değiştir

```
document.getElementById('camera-btn').addEventListener('click', async () => {  
  //Check if what the state of muted currently is  
  //Disable button  
  if(!localTrackState.videoTrackMuted){  
    //Mute your audio  
    await localTracks.videoTrack.setMuted(true);  
    localTrackState.videoTrackMuted = true  
    document.getElementById('camera-btn').style.backgroundColor ='rgb(255,  
80, 80, 0.7)'  
  }else{  
    await localTracks.videoTrack.setMuted(false)  
    localTrackState.videoTrackMuted = false  
    document.getElementById('camera-btn').style.backgroundColor ='#1f1f1f8e'  
  }  
}
```



```
})
```



Kamera açma ve kapatma olayı(event)

Ses açma ve kapatma matnığı ile aynıdır.

```
await localTracks.videoTrack.setMuted(true);
```

join basıldıktan sonra

```
videoTrack = AgoraRTC.createCameraVideoTrack()
```

olacak .createCameraVideoTrack() Agoradan getirilen metod bu yüzden setMuted metodu kullanabiliriz.

videoTrack başladığımız değer =null

```
document.getElementById('leave-btn').addEventListener('click', async () => {
    //Loop threw local tracks and stop them so unpublish event gets triggered,
    then set to undefined
    //Hide footer
    for (trackName in localTracks){
        let track = localTracks[trackName]

        if(track){
            track.stop()
            track.close()
            localTracks[trackName] = null
        }
    }
})
```

```
//Leave the channel
  await client.leave()
  document.getElementById('footer').style.display = 'none'
  document.getElementById('user-streams').innerHTML = ''
  document.getElementById('join-wrapper').style.display = 'block'
})
```

Leave butonu olayı(event)

For iki defa dönecek

Birincisinde trackName=audioTrack

Ikincisinde ise trackName=videoTrack

localTracks daha önce oluşturulan nesne

eğer track!=null ise if e gir

stop ve close metodlar agoradan çağırılıyor

video ve ses durdurup kapatılıyor

kullanıcı çıkana kadar bekle() (await vasıtası ile)

leave() metodu agorada çağırılıyor

son üç satır Html sayfası ilk haline dön

```
let joinStreams = async () => {
```

join butonuna basıldıktan sonra kullanılan fonksiyon(Önemli)

```
client.on("user-published", handleUserJoined);
```

Agoradan kullanılan olay(event)

“user-published” olay ismi

handleUserJoined: kullanılacak fonksiyon(daha sonra anlatılacak)

Agora API Referene

user-published

- `user-published(user: IAgoraRTCRemoteUser, mediaType: "audio" | "video"): void`

Occurs when a remote user publishes an audio or video track.

You can subscribe to and play the audio or video track in this callback. See [subscribe](#) and [RemoteTrack.play](#).

The SDK also triggers this callback to report the existing tracks in the channel when a user joins the channel.

```
client.on("user-published", async (user, mediaType) => {  
  await client.subscribe(user, mediaType);  
  if (mediaType === "video") {  
    console.log("subscribe video success");  
    user.videoTrack.play("xxx");  
  }  
  if (mediaType === "audio") {  
    console.log("subscribe audio success");  
    user.audioTrack.play();  
  }  
})
```

Parameters

- **user:** *IAgoraRTCRemoteUser*

Information of the remote user.

- **mediaType:** *"audio" | "video"*

Type of the track.

- **"audio"** : The remote user publishes an audio track.
- **"video"** : The remote user publishes a video track.

Returns *void*

```
client.on("user-left", handleUserLeft);
```

Agoradan kullanılan olay(event)

“user-left” olay ismi

handleUserLeft: kullanılacak fonksiyon(daha sonra anlatılacak)

Agora API Referene :

user-left

- `user-left(user: IAgoraRTCRemoteUser, reason: string): void`

Occurs when a remote user becomes offline.

The SDK triggers this callback when one of the following situations occurs:

- A remote user calls `leave` and leaves the channel.
- A remote user has dropped offline. If no data packet of the user or host is received for 20 seconds, the SDK assumes that the user has dropped offline. A poor network connection may cause a false positive.
- A remote user switches the client role from host to audience.

In live-broadcast channels, the SDK triggers this callback only when a host goes offline.

Parameters

- **user:** *IAgoraRTCRemoteUser*

Information of the user who is offline.

- **reason:** *string*

Reason why the user has gone offline.

- `"Quit"` : The user calls `leave` and leaves the channel.
- `"ServerTimeOut"` : The user has dropped offline.
- `"BecomeAudience"` : The client role is switched from host to audience.

Returns *void*

```
client.enableAudioVolumeIndicator(); // Triggers the "volume-indicator" callback
event every two seconds.
client.on("volume-indicator", function(evt){
    for (let i = 0; evt.length > i; i++){
        let speaker = evt[i].uid
        let volume = evt[i].level
        if(volume > 0){
            document.getElementById(`volume-${speaker}`).src =
'./assets/volume-on.svg'
```

```

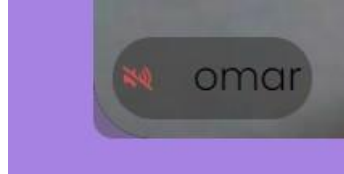
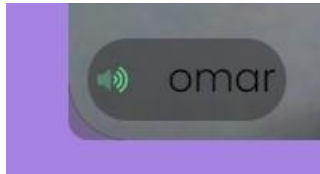
        }else{
            document.getElementById(`volume-${speaker}`).src =
'./assets/volume-off.svg'
        }

    }

});

```

Tüm kullanıcılardan ses test eden kod her kullanıcı için sesi açık ya da kapalı ise gösteriyor



Agora API Reference

enableAudioVolumeIndicator

• enableAudioVolumeIndicator(): void

Enables the volume indicator.

This method enables the SDK to regularly report the remote users who are speaking and their volumes.

After the volume indicator is enabled, the SDK triggers the [AgoraRTCClient.on\("volume-indicator"\)](#) callback to report the volumes every two seconds, regardless of whether there are active speakers in the channel.

```

client.enableAudioVolumeIndicator();
client.on("volume-indicator", volumes => {
    volumes.forEach((volume, index) => {
        console.log(`${index} UID ${volume.uid} Level ${volume.level}`);
    });
});

```

Returns void

volume-indicator

- `volume-indicator(result: object[]): void`

Reports all the speaking remote users and their volumes.

It is disabled by default. You can enable this callback by calling [enableAudioVolumeIndicator](#). If enabled, it reports the users' volumes every two seconds regardless of whether there are users speaking.

The volume is an integer ranging from 0 to 100. Usually a user with volume above five is a speaking user.

```
client.on("volume-indicator", function(result){
  result.forEach(function(volume, index){
    console.log(`${index} UID ${volume.uid} Level ${volume.level}`);
  });
});
```

Parameters

- `result: object[]`

Returns `void`

```
[config.uid, localTracks.audioTrack, localTracks.videoTrack] =
await Promise.all([
  client.join(config.appid, config.channel, config.token || null, config.uid
|| null),
  AgoraRTC.createMicrophoneAudioTrack(),
  AgoraRTC.createCameraVideoTrack()
])
```

Yukarıdaki kodu açarsak eğer

```
config.uid = client.join(config.appid, config.channel, config.token || null,
  config.uid || null)
```

```
localTracks.audioTrack = AgoraRTC.createMicrophoneAudioTrack()
```

```
localTracks.videoTrack = AgoraRTC.createCameraVideoTrack()
```

Promise.all javascriptte kullanılan metod

```
let player = `

Kullanıcıyı göstermek için gerekli Html kodu



Classlar css fileden alındı



```
document.getElementById('user-streams').insertAdjacentHTML('beforeend', player);
```



user-streams: html sayfaında da bir <div> ona player kodlar eklemek için kullanılan kod



```
Player user stream in div
localTracks.videoTrack.play(`stream-${config.uid}`)
```



player değişende son div video ona ekle



Agora API Referance


```


play

• `play(element: string | HTMLElement, config?: VideoPlayerConfig): void`

Overrides [ITrack.play](#)

Plays a remote video track on the web page.

Parameters

▪ **element:** `string | HTMLElement`

Specifies a DOM element. The SDK will create a `<video>` element under the specified DOM element to play the video track. You can specify a DOM element in either of following ways:

- `string` : Specify the ID of the DOM element.
- `HTMLElement` : Pass a DOM object.

▪ **Optional config:** `VideoPlayerConfig`

Sets the playback configurations, such as display mode and mirror mode. See [VideoPlayerConfig](#). By default, the SDK enables mirror mode for a local video track.

Returns `void`

```
await client.publish([localTracks.audioTrack, localTracks.videoTrack])
```

kullanıcının sesi ve videosu yayınla

Agora Api Referance

```
• publish(tracks: ILocalTrack | ILocalTrack[]): Promise<void>
```

Publishes local audio and/or video tracks to a channel.

After publishing the local tracks, the `AgoraRTCClient.on("user-published")` callback is triggered on the remote client.

Note:

- In an interactive live streaming, call `setClientRole` to set the user role as the host before calling this method.
- You can call this method multiple times to add tracks for publishing.
- An `AgoraRTCClient` object can publish multiple audio tracks. The SDK automatically mixes the audio tracks into one audio track. Exception: Safari does not support publishing multiple audio tracks on versions earlier than Safari 12.
- An `AgoraRTCClient` object can publish **only one video track**. If you want to switch the published video track, for example, from a camera video track to a scree-sharing video track, you must unpublish the published video track.

Parameters

- tracks: *ILocalTrack* | *ILocalTrack*[]

Local tracks created by `AgoraRTC.createMicrophoneAudioTrack` / `AgoraRTC.createCameraVideoTrack` or other methods.

```
let handleUserJoined = async(user, mediaType) => {
```

“user-published” olayı (event)için kullanılan fonksiyon(Önemli)

user ve mediaType değerleri Agoradan gelir

```
remoteTracks[user.uid] = user
```

daha önce oluşturulan remoteTracks nesnesine yeni gelen kullanıcı ekler

nasıl eklendiğini aşağıdaki javascript kod anlatıyor

```
> isim = {}  
< ▶ {}  
  
> isim["Araba"] = "Audi"  
< 'Audi'  
  
> isim["sehir"] = "Elazig"  
< 'Elazig'  
  
> isim  
< ▶ {Araba: 'Audi', sehir: 'Elazig'}  
  
> isim.Araba  
< 'Audi'  
  
> isim.sehir  
< 'Elazig'  
  
> |
```

```
await client.subscribe(user, mediaType)
```

kullanıcı onyala

Agora API Referance

subscribe

• subscribe(user: [IAgoraRTCRemoteUser](#), mediaType: "video"): Promise<[IRemoteVideoTrack](#)>

Subscribes to the audio and/or video tracks of a remote user.

```
await client.subscribe(user, "audio");
user.audioTrack.play();
```

Parameters

▪ user: [IAgoraRTCRemoteUser](#)

The remote user.

▪ mediaType: "video"

The media type of the tracks to subscribe to.

- "video" : Subscribe to the video track only.
- "audio" : Subscribe to the audio track only.

Returns Promise<[IRemoteVideoTrack](#)>

When the subscription succeeds, the SDK adds the subscribed tracks to [user.audioTrack](#) and [user.videoTrack](#). You can go on to call [audioTrack.play](#) or [videoTrack.play](#) to play the tracks.

The Promise object throws the TRACK_IS_NOT_PUBLISHED error if the specified tracks do not exist.

• subscribe(user: [IAgoraRTCRemoteUser](#), mediaType: "audio"): Promise<[IRemoteAudioTrack](#)>

• subscribe(user: [IAgoraRTCRemoteUser](#), mediaType: "video" | "audio"): Promise<[IRemoteTrack](#)>

```
if (mediaType === 'video'){
  let player = document.getElementById(`video-wrapper-${user.uid}`)
  if (player !== null){
    player.remove()
  }
}
```

Eğer mediyeye type video ise agoradan gelen user player değişkenine html'deki arayüzü getir, eğer daha önce eklenmiş ise yani null değil ise onu html sayfasından kaldır çünkü daha sonar gelecek kodlar yineden

ekleyecek (palyer.remove olmasaydı video buttona iki kez bastığımız zaman aynı kullanıcıya ait iki arayüzü açacaktı

```
player = `

yukarıdaki kod daha önce onun gibi kullanıcı arasındaki fark  
biri{config.uid} bu ise {user.uid}



{config.uid} bizim bilgisayarımız



{user.uid} diğer kullanıcılar user agoradan gelen bir parameter



```
if (mediaType === 'audio') {
 user.audioTrack.play();
}
```



Eğer mediaType eşittir 'audio' gelen kullanıcı sesi çalıştır



```
let handleUserLeft = (user) => {

 //Remove from remote users and remove users video wrapper
 delete remoteTracks[user.uid]

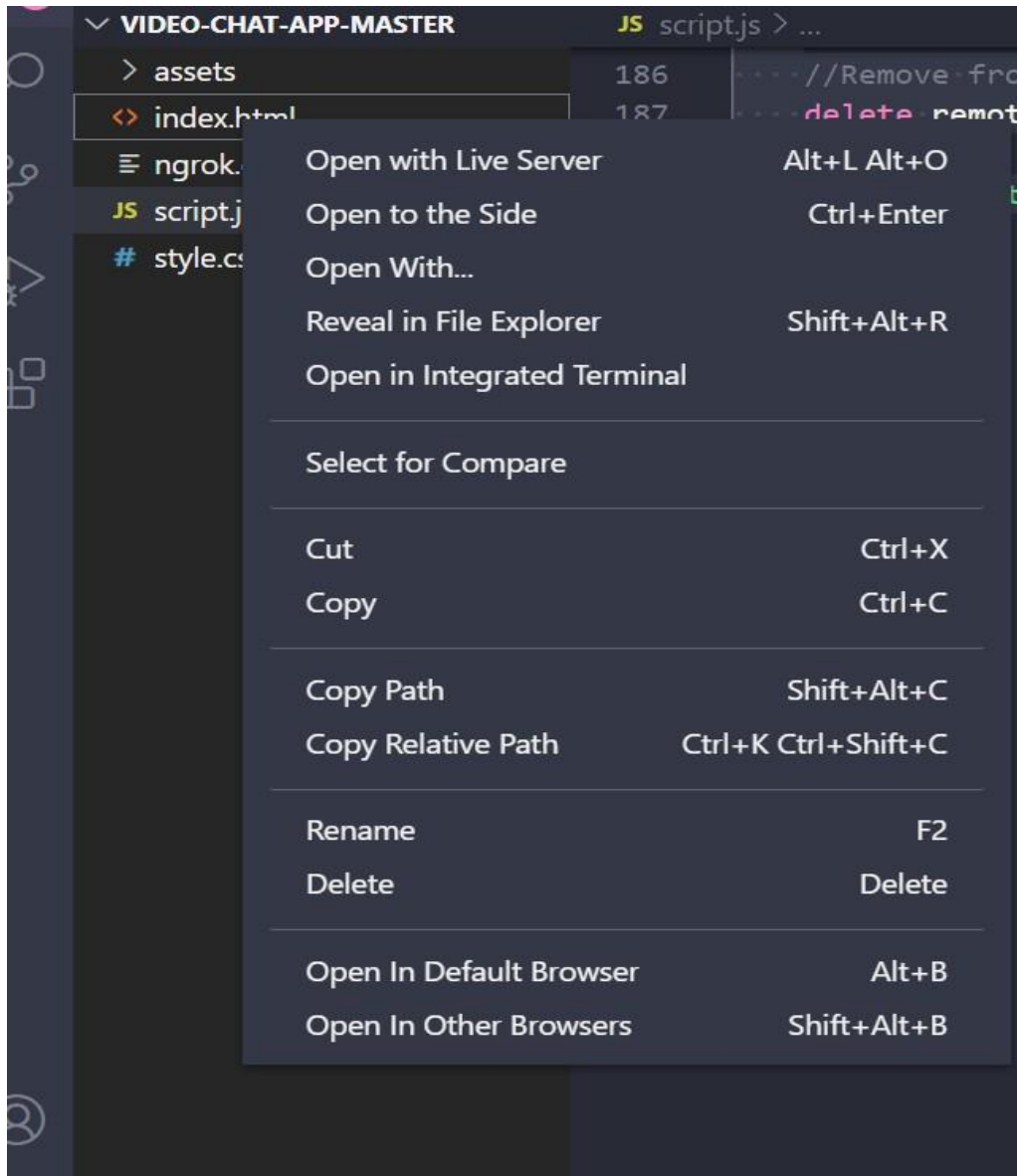
 document.getElementById(`video-wrapper-${user.uid}`).remove()
}
```


```

Bir kullanıcı çıktığı zaman onu sil ve arayüzünü kaldır.

app nasıl çalıştırılır?

Vs code kullanarak Html sayfası üzerinde sağ tıkla ve Open with Live Server seçiyoruz.



Çıkacak arayüz:



Local host olduğu için birden fazla kullanıcı sadece aynı bilgisayar üzerinde çalıştırabiliyoruz(browser üzerinde birden fazla pencere)

Birden fazla cihaz üzerinde çalıştırmak için bir Url ihtiyacı vardır.

Bunun için ngrok kullanmak gerekiyor.

<https://ngrok.com/> web sitesinde hesap oluşturmak gerekiyor

daha sonra ngrok indirmek gerek ve App'in aynı klasörü yapıştırmak gerek.

daha sonra vs code'da new Terminal açıp ve ngrok sayfasında gösterdiği kod yapıştırmak gerekiyor(./ngrok http (Port numarası))

Yalnız vs code hangi port kullandığını dikkat etmek gerek.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\batte\Desktop\video-chat-app-master> ./ngrok http 5500
```

Daha sonra link çıkacak link her hangi bir cihazda çalışır.