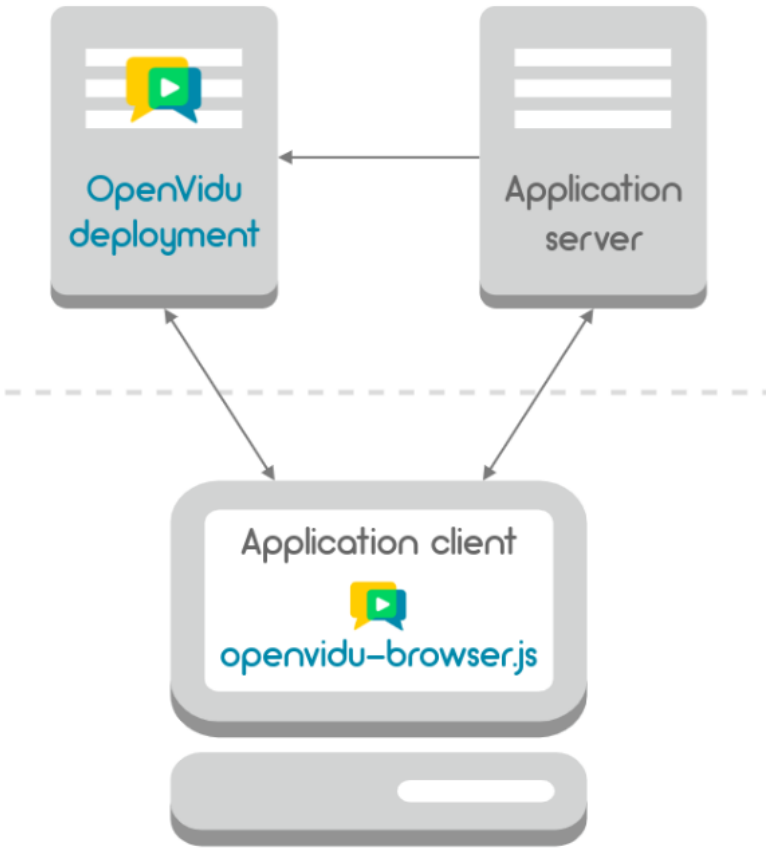
OpenVidu

OpenVidu

- OpenVidu는 웹 또는 모바일 애플리케이션에 화상 통화를 쉽게 추가할 수 있는 플랫폼입니다.
- 애플리케이션에 매우 쉽게 통합할 수 있는 완전한 기술 스택을 제공합니다.
- 주요 목표는 개발자가 코드에 미치는 영향을 최소화하면서 매우 빠르게 앱에 실시간 커뮤니케이션을 추가할 수 있도록 하는 것입니다.

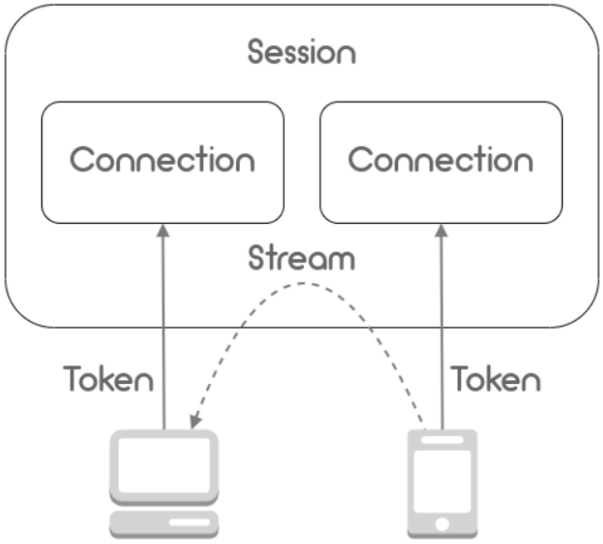
- OpenVidu 애플리케이션 아키텍쳐에서 필요한 3가지 컴포넌트
  - OpenVidu deployment
    - 실시간 오디오 및 비디오 스트리밍에 필요한 모든 인프라 제공 (블랙박스)
    - 애플리케이션에서 배포하고 사용하기만 하면 된다.
  - server application
    - 사용자의 application server에서 실행되며, OpenVidu deployment에서 제공하는 REST API를 사용 (모든 REST 클라이언트로 REST API endpoints를 직접 호출할 수 있음)
    - 이러한 방식으로 세션, 연결을 생성하고 화상 통화를 안전하게 관리할 수 있다.
  - client application
    - 웹 브라우저, 모바일 장치 또는 데스크톱 애플리케이션에서 실행됨
    - openvidu-browser.js SDK를 사용하여 OpenVidu 배포와 통신하고, 세션에 연결하고, 미디어 스트림을 게시 및 구독하고, 클라이언트 측에서 화상 통화의 다른 측면을 관리



기본 개념

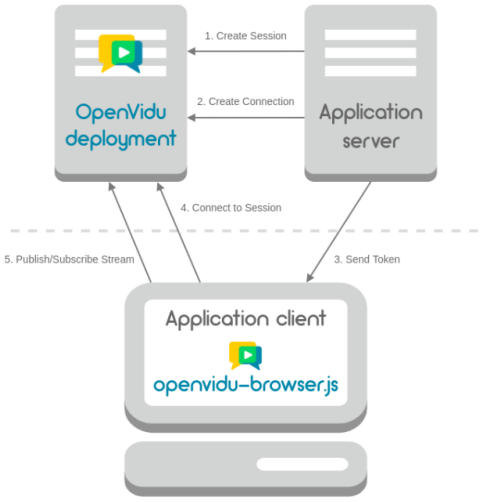
- session
  - 세션은 참가자가 연결하여 오디오 및 비디오 스트림을 보내고 받을 수 있는 가상 공간
  - 동일한 세션에 연결된 참가자만 서로 들을 수 있다.
  - 세션당 참가자 수는 사용사례에 따라 다릅니다.
- Connection
  - 연결은 세션의 각 참가자를 나타냄
  - 세션에 연결하려면 응용 프로그램 서버에서 초기화해야 한다.
  - 해당 토큰을 응용 프로그램 클라이언트에 전달해야 한다.
  - 클라이언트가 해당 슬롯을 차지할 수 있으며, 클라이언트가 연결을 점유하면 세션의 참가자로 간주
- Token
  - 참가자에게 세션에 대한 액세스 권한을 부여하려면 토큰 필요
  - 각 참가자는 세션에 연결할 때 토큰을 사용
  - Token은 Connection과 항상 연관된다.
    - 사용자의 application client에 전달할 토큰을 얻으려면, 사용자의 application server에서 Connection을 생성해야 한다.
  - 토큰은 해당 연결의 슬롯을 점유할 수 있도록 하는 키로 볼 수 있는 연결의 속성일뿐
  - application client에서 사용될 때, 토큰은 세션 내부의 참가자에게 metadata 그리고 특정 기능을 제공할 수 있다.
- Stream

- 스트림은 세션으로 흐르는 미디어 스트림이다.
  - 참가자는 스트림을 게시할 수 있으며
  - 동일한 세션의 다른 참가자는 그것을 subscribe 할 수 있다.
- 세션에서 게시할 수 있는 방법, 시기, 사람 및 유형에 대한 제한은 없다.



Workflow of an OpenVidu Session

1. 사용자의 application server에서 세션을 초기화합니다.
2. application server에서 세션에 대한 **Connection을 생성**합니다.
   참가자가 세션에 있을 만큼 많은 연결을 만들어야 합니다.
   각 연결에는 세션에 대한 단일 액세스를 제공하는 토큰이 있습니다.
3. 세션에 연결할 수 있도록 **클라이언트**에 토큰을 전달합니다.
4. 각 **클라이언트**는 토큰을 사용하여 openvidu-browser.js를 사용하여 **세션에 연결**합니다. 성공적으로 완료하면 세션의 참가자로 간주됩니다.
5. 세션에 연결되면 참가자는 openvidu-browser.js를 사용하여 스트림을 게시할 수 있습니다. 세션의 다른 모든 참가자는 구독할 기회를 갖게 됩니다.



OpenVidu in your application server

- application server에 OpenVidu를 통합하는 것은 화상 통화를 안전하게 만드는 데 필수적입니다.

OpenVidu in your application client

- application client에 OpenVidu를 통합하는 다양한 옵션이 있다.
- 고려해야 할 단순성과 사용자 정의 사이에 투쟁이 있기 때문에 올바른 옵션을 선택하는 것이 중요
- OpenVidu는 클라이언트 측에 고유한 대안을 제공하려고 노력하므로 특정 사용 사례와 클라이언트 측에 투자하려는 리소스에 따라 가장 적합한 것을 선택할 수 있습니다.

- Ready-to-use component

  A Web Component ready to be inserted in your application

  - Props
    - 몇 줄의 코드로 화상 회의 가능
  - Cons
    - customization의 한계
    - desktop browsers 용으로 특별히 설계되었으며, 다른 플랫폼에는 불일치가 있을 수 있다.

- OpenVidu Components

  자체 application build를 위한 화상 회의 구성 요소를 제공하는 frontedn library

  - props
    - 매우 유연한 컴포넌트 : 모든 component의 요소를 조정, 확제 또는 교체
    - 최신 기능으로 클라이언트 코드를 쉽게 최신상태로 유지
  - cons
    - Angular에만 사용 가능

- Full control of the UI

  클라이언트 SDK를 사용하여 웹, 모바일 또는 데스크톱 애플리케이션에서 처음부터 OpenVidu 통합

  - props
    - 무제한 수준의 사용자 정의 : 처음부터 원하는 대로 자신만의 UI를 구축
    - 모든 클라이언트 플랫폼에서 사용 가능 : 데스크톱 및 모바일 브라우저, 기분 애플리케이션 …
  - cons
    - 학습 곡선을 원활하게 하기 위한 많은 자습서가 있지만 더 높은 복잡성

▼ npm. start시 에러 발생하는 경우

Error: error:0308010c:digital envelope routines::unsupported [Node Error Solved]

If you work with Node.js and command line interface solutions like Webpack, create-react-app, or vue-cli-service, you might have encountered the error, Error: error:0308010c:digital envelope routines::unsupported. You're not alone, because I'm currently getting it too: The

https://www.freecodecamp.org/news/error-error-0308010c-digital-envelope-routines-unsupported-node-error-solved/

```
// package.json 파일scripts 수정

"scripts": {
    ...
    "start": "react-scripts --openssl-legacy-provider start",
    ...
  },
```

- OpenVidu 애플리케이션 아키텍쳐에서 필요한 3가지 컴포넌트
  - OpenVidu deployment
  - server application
  - client application

- Docker 설치
- Run OpenVidu deployment

```
docker run -p 4443:4443 --rm -e OPENVIDU_SECRET=MY_SECRET openvidu/openvidu-dev:2.28.0
```

React

>>> App.js

```
import { OpenVidu } from 'openvidu-browser';

// These properties are in the state's component in order to re-render the HTML whenever their values change
this.state = {
```

```
    mySessionId: 'SessionA',
    myUserName: 'Participant' + Math.floor(Math.random() * 100),
    session: undefined,
    mainStreamManager: undefined, // Main video of the page. Will be the 'publisher' or one of the 'subscribers'
    publisher: undefined,
    subscribers: [],
};
```

- OpenVidu 개체를 사용하면 바로 뒤에 선언되는 Session 개체를 얻을 수 있습니다.
- 게시자는 자체 로컬 웹캠 스트림이 되고 구독자 배열은 화상 통화에서 다른 사용자의 활성 스트림을 저장합니다.
- 마지막으로, mySessionId 및 myUserName 매개변수는 곧 보게 되겠지만 화상 통화와 참가자의 닉네임을 나타냅니다.
- 사용자가 App.js 템플릿에 정의된 제출 입력을 클릭할 때마다 joinSession() 메서드가 호출됩니다.
    - 먼저 OpenVidu 개체를 가져오고 상태에서 세션 속성을 초기화합니다.

```
// --- 1) Get an OpenVidu object ---

this.OV = new OpenVidu();

// --- 2) Init a session ---

this.setState(
    {
        session: this.OV.initSession(),
    },
    () => {
        // See next step
    }
);
```

    - 그런 다음 관심 있는 세션 이벤트를 구독합니다.
        - React를 사용하고 있으므로 원격 미디어 스트림을 관리하는 좋은 접근 방식은 각 구독자 객체에 공통 구성 요소를 공급하고 비디오를 관리하도록 하는 것
        - 이 컴포넌트는 UserVideoComponent가 됨.
        - 이렇게 하면 수신한 각각의 새 구독자를 구독자 배열에 저장하고 필요할때마다 삭제된 모든 구독자를 제거해야 함

```
var mySession = this.state.session;

// --- 3) Specify the actions when events take place in the session ---

// On every new Stream received...
mySession.on('streamCreated', (event) => {
    // Subscribe to the Stream to receive it. Second parameter is undefined
    // so OpenVidu doesn't create an HTML video by its own
    var subscriber = mySession.subscribe(event.stream, undefined);

    //We use an auxiliar array to push the new stream
    var subscribers = this.state.subscribers;

    subscribers.push(subscriber);

    // Update the state with the new subscribers
    this.setState({
        subscribers: subscribers,
    });
});

    // On every Stream destroyed...
mySession.on('streamDestroyed', (event) => {
    event.preventDefault();

    // Remove the stream from 'subscribers' array
    this.deleteSubscriber(event.stream.streamManager);
```

```
});

// On every asynchronous exception...
mySession.on('exception', (exception) => {
    console.warn(exception);
});

// See next step
```

1. `var mySession = this.state.session;`
    - `mySession` 변수에 현재 세션 객체를 저장합니다. 세션은 OpenVidu 서버와의 연결을 나타내며, 스트림을 생성하고 관리하는 데 사용됩니다.
2. `mySession.on('streamCreated', (event) => { ... });`
    - 'streamCreated' 이벤트가 발생할 때마다 실행되는 코드 블록입니다. 이 이벤트는 새로운 스트림이 생성되었음을 나타냅니다.
    - 코드에서는 해당 스트림을 구독하고, `subscribers` 배열에 구독자를 추가한 후 상태를 업데이트합니다.
3. `mySession.on('streamDestroyed', (event) => { ... });`
    - 'streamDestroyed' 이벤트가 발생할 때마다 실행되는 코드 블록입니다. 이 이벤트는 스트림이 파괴되었음을 나타냅니다.
    - 코드에서는 `deleteSubscriber` 함수를 호출하여 해당 스트림을 `subscribers` 배열에서 제거합니다.
4. `mySession.on('exception', (exception) => { ... });`
    - 'exception' 이벤트가 발생할 때마다 실행되는 코드 블록입니다. 이 이벤트는 비동기 예외가 발생했음을 나타냅니다.
    - 현재 예외를 콘솔에 기록합니다.

```
{this.state.subscribers.map((sub, i) => (
    <div key={i} className="stream-container col-md-6 col-xs-6">
        <UserVideoComponent streamManager={sub} mainVideoStream={this.handleMainVideoStream} />
    </div>
))}
```

- **Get an OpenVidu toke**
    - 세션에 참여할 준비가 되었습니다.
    - 그러나 액세스 권한을 얻으려면 여전히 토큰이 필요하므로 서버 응용 프로그램에 토큰을 요청합니다.
    - 그러면 서버 애플리케이션이 OpenVidu 배포에 대한 토큰을 요청합니다.

```
// --- 4) Connect to the session with a valid user token ---

// Get a token from the OpenVidu deployment
this.getToken().then((token) => {
    // See next point to see how to connect to the session using 'token'
}
```

    - 이것은 애플리케이션 서버에서 최종적으로 토큰을 검색하는 역할을 하는 코드입니다. 튜토리얼은 axios 라이브러리를 사용하여 필요한 HTTP 요청을 수행합니다.

```
async getToken() {
    const sessionId = await this.createSession(this.state.mySessionId);
```

```
    return await this.createToken(sessionId);
}

async createSession(sessionId) {
    const response = await axios.post(APPLICATION_SERVER_URL + 'api/sessions', { customSessionId: sessionId }, {
        headers: { 'Content-Type': 'application/json', },
    });
    return response.data; // The sessionId
}

async createToken(sessionId) {
    const response = await axios.post(APPLICATION_SERVER_URL + 'api/sessions/' + sessionId + '/connections', {}, {
        headers: { 'Content-Type': 'application/json', },
    });
    return response.data; // The token
```

1. `getToken()` 함수가 호출되면 다음과 같은 일련의 작업이 발생합니다:
    - `this.createSession(this.state.mySessionId)`를 호출하여 세션을 생성합니다. 이때 `mySessionId`는 세션의 사용자 정의 ID입니다.
    - `createSession()` 함수에서 반환된 세션 ID를 `sessionId` 변수에 저장합니다.
    - `this.createToken(sessionId)`를 호출하여 해당 세션에 대한 토큰을 생성합니다.
    - `createToken()` 함수에서 반환된 토큰을 `getToken()` 함수의 결과로 반환합니다.
2. `createSession(sessionId)` 함수는 다음과 같은 일을 수행합니다:
    - `axios.post()`를 사용하여 OpenVidu 서버의 세션 API 엔드포인트로 POST 요청을 보냅니다.
    - 요청 본문에는 `customSessionId`라는 키와 `sessionId` 값을 가진 JSON 객체가 포함됩니다.
    - OpenVidu 서버는 이 요청을 처리하고 새 세션을 생성합니다.
    - 생성된 세션 ID가 응답 데이터인 `response.data`로 반환됩니다.
3. `createToken(sessionId)` 함수는 다음과 같은 일을 수행합니다:
    - `axios.post()`를 사용하여 OpenVidu 서버의 연결 생성 API 엔드포인트로 POST 요청을 보냅니다.
    - 요청 URL에는 이전에 생성한 세션 ID가 포함됩니다.
    - OpenVidu 서버는 이 요청을 처리하고 해당 세션에 대한 새 연결 및 토큰을 생성합니다.
    - 생성된 토큰이 응답 데이터인 `response.data`로 반환됩니다.

이러한 절차를 통해 `getToken()` 함수는 OpenVidu 서버로부터 세션 및 토큰을 받아오게 됩니다. 이후 애플리케이션에서는 이 토큰을 사용하여 해당 세션에 대한 인증을 수행하고 영상 및 오디오 스트림을 전송하고 수신할 수 있습니다.

- **Finally connect to the session and publish your webcam**

```
// --- 4) Connect to the session with a valid user token ---

// Get a token from the OpenVidu deployment
this.getToken().then((token) => {
    // First param is the token got from the OpenVidu deployment. Second param can be retrieved by every user on event
    // 'streamCreated' (property Stream.connection.data), and will be appended to DOM as the user's nickname
    mySession.connect(token, { clientData: this.state.myUserName })
        .then(async () => {

            // --- 5) Get your own camera stream ---

            // Init a publisher passing undefined as targetElement (we don't want OpenVidu to insert a video
            // element: we will manage it on our own) and with the desired properties
            let publisher = await this.OV.initPublisherAsync(undefined, {
                audioSource: undefined, // The source of audio. If undefined default microphone
                videoSource: undefined, // The source of video. If undefined default webcam
                publishAudio: true, // Whether you want to start publishing with your audio unmuted or not
                publishVideo: true, // Whether you want to start publishing with your video enabled or not
                resolution: '640x480', // The resolution of your video
                frameRate: 30, // The frame rate of your video
```

```
                insertMode: 'APPEND', // How the video is inserted in the target element 'video-container'
                mirror: false, // Whether to mirror your local video or not
            });

            // --- 6) Publish your stream ---

            mySession.publish(publisher);

            // Obtain the current video device in use
            var devices = await this.OV.getDevices();
            var videoDevices = devices.filter(device => device.kind === 'videoinput');
            var currentVideoDeviceId = publisher.stream.getMediaStream().getVideoTracks()[0].getSettings().deviceId;
            var currentVideoDevice = videoDevices.find(device => device.deviceId === currentVideoDeviceId);

            // Set the main video in the page to display our webcam and store our Publisher
            this.setState({
                currentVideoDevice: currentVideoDevice,
                mainStreamManager: publisher,
                publisher: publisher,
            });
        })
        .catch((error) => {
            console.log('There was an error connecting to the session:', error.code, error.message);
        });
});
```

▼ chat gpt 해설

위의 코드는 OpenVidu를 사용하여 React 애플리케이션에서 세션에 연결하는 과정을 보여줍니다. 아래에서 코드를 단계별로 설명하겠습니다.

단계 4: 유효한 사용자 토큰으로 세션에 연결

```
javascriptCopy code
this.getToken().then((token) => {
    mySession.connect(token, { clientData: this.state.myUserName })
        .then(async () => {
            // ...
        })
        .catch((error) => {
            console.log('There was an error connecting to the session:', error.code, error.message);
        });
});
```

- `getToken()` 함수를 호출하여 OpenVidu 배포에서 토큰을 가져옵니다. 토큰은 현재 사용자를 식별하는 데 사용됩니다.
- `mySession.connect()` 함수를 호출하여 획득한 토큰과 옵션을 전달하여 세션에 연결합니다. `clientData`는 사용자의 닉네임을 DOM에 추가하는 데 사용됩니다.
- 연결에 성공하면 `then` 블록이 실행됩니다. 이후의 코드는 세션에 연결한 후 수행할 작업을 정의합니다.
- 연결에 실패하면 `catch` 블록이 실행되고 오류 메시지가 콘솔에 출력됩니다.

단계 5: 자신의 카메라 스트림 가져오기

```
javascriptCopy code
let publisher = await this.OV.initPublisherAsync(undefined, {
    audioSource: undefined,
    videoSource: undefined,
    publishAudio: true,
    publishVideo: true,
    resolution: '640x480',
    frameRate: 30,
    insertMode: 'APPEND',
    mirror: false,
});
```

- `this.OV.initPublisherAsync()` 함수를 호출하여 자신의 카메라 스트림을 가져옵니다. 이 함수는 비동기 함수이므로 `await`를 사용하여 호출합니다.
- `undefined`로 설정된 `targetElement` 매개변수는 OpenVidu에게 비디오 요소를 자동으로 삽입하지 않도록 지정합니다. 대신 개발자가 직접 관리할 것입니다.
- `audioSource` 및 `videoSource`를 `undefined`로 설정하면 기본 마이크와 웹캠을 사용합니다.
- `publishAudio` 및 `publishVideo`를 `true`로 설정하면 오디오와 비디오를 활성화한 채로 스트림을 게시합니다.
- `resolution`은 비디오 해상도를 설정합니다.
- `frameRate`는 비디오의 프레임 속도를 설정합니다.
- `insertMode`는 비디오를 삽입할 대상 요소인 'video-container'에 대한 설정입니다.
- `mirror`를 `false`로 설정하면 로컬 비디오를 반전시키지 않습니다.

단계 6: 스트림 게시하기

```
javascriptCopy code
mySession.publish(publisher);
```

- `mySession.publish()` 함수를 호출하여 생성한 스트림을 게시합니다. 이를 통해 다른 참여자들이 현재 사용자의 비디오 및 오디오를 볼 수 있게 됩니다.

나머지 코드는 현재 사용자의 비디오 장치를 가져와서 화면에 표시하고, 상태를 업데이트하여 사용자의 Publisher를 저장하는 작업을 수행합니다. 이는 애플리케이션에서 자체적으로 관리하는 내용입니다.

위의 코드는 React를 사용하여 OpenVidu 세션에 연결하고 자신의 비디오 스트림을 게시하는 데 필요한 주요 단계를 보여줍니다.

- 세션 초기화
- 세션 생성

1. `createSession(sessionId)` 함수는 OpenVidu 서버로 세션 ID를 요청합니다.
   - OpenVidu 서버는 세션 ID를 확인하고 응답합니다.
   - 반환된 세션 ID는 클라이언트로 반환됩니다.
2. `createToken(sessionId)` 함수는 OpenVidu 서버로 세션 ID를 전달하여 토큰을 요청합니다.
   - OpenVidu 서버는 전달받은 세션 ID를 확인하고 해당 세션에 대한 토큰을 생성합니다.
   - 생성된 토큰은 클라이언트로 반환됩니다.

즉, 코드는 세션 ID를 생성하고 OpenVidu 서버에 전달하여 서버 측에서 세션 및 토큰을 생성하고 클라이언트로 반환하도록 동작합니다. 이렇게 생성된 토큰은 클라이언트 측에서 OpenVidu에 접속하고 세션에 참여하기 위해 사용됩니다.