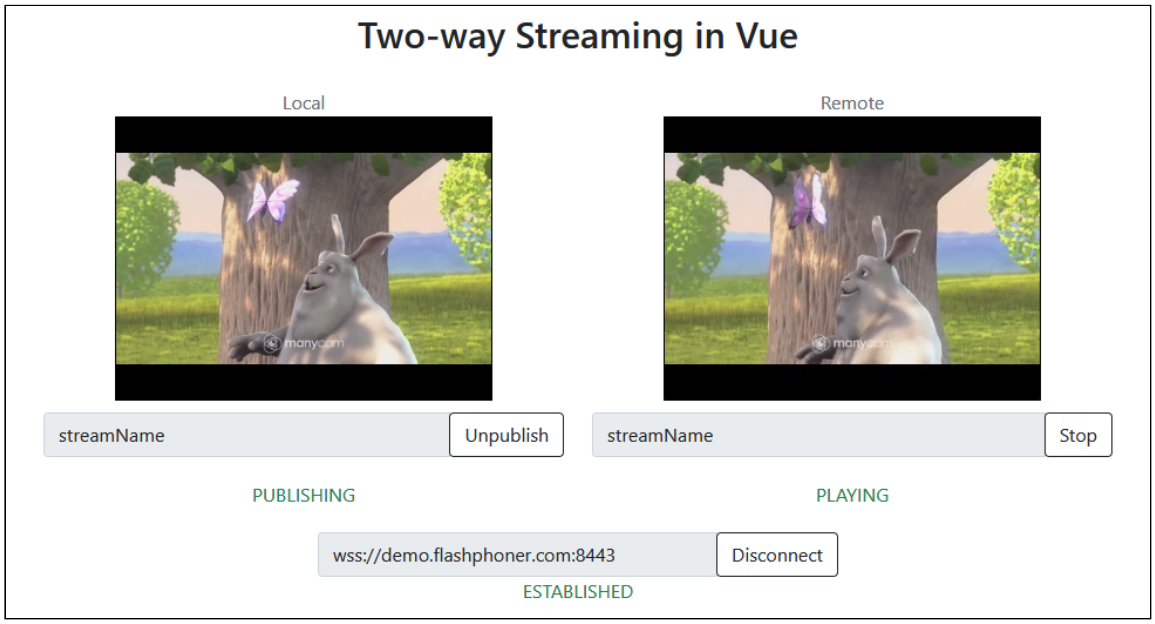
Two Way Streaming Vue

Описание

Приложение Two Way Streaming React показывает пример использования Web SDK во Vue.js приложении для публикации и проигрывания WebRTC потока



Проект доступен на GitHub, а также в архивах сборок Web SDK по следующему пути examples/react/two-way-streaming-vue, начиная со сборки 2.0.202.

Сборка проекта

1. Загрузите исходные тексты WebSDK

```
git clone https://github.com/flashphoner/flashphoner_client.git
```

2. Перейдите в каталог примера

```
cd flashphoner_client/examples/vue/two-way-streaming-vue
```

3. Установите зависимости

```
npm install
```

4. Запустите сборку для локального тестирования

```
npm run serve
```

или для выгрузки на свой веб-сервер

```
npm run build
```

Работа с кодом примера

Для разбора исходного кода примера возьмем версию с хэшем `6dc44b8`, которая доступна здесь

Код приложения расположен в файле TwoWayStreamingApp.vue, дополнительные функции в файле fp-utils.js

1. Импорт API

code

```
import * as FPUtils from './fp-utils.js';
import * as Flashphoner from '@flashphoner/websdk';
```

2. Инициализация API

`Flashphoner.init()` code

```
onLoad() {
   try {
     Flashphoner.init({});
     ...
   }
   catch(e) {
     console.log(e);
     ...
   }
}
```

3. Подключение к серверу и получение события, подтверждающего установку соединения

`Flashphoner.createSession()`, `SESSION_STATUS.ESTABLISHED` code

```
onConnectClick() {
   let url = this.serverUrl;
```

```
    let session = this.session;

    if (!session) {
      console.log("Create new session with url " + url);
      this.connectButtonDisabled = true;
      this.serverUrlDisabled = true;
      Flashphoner.createSession({urlServer:
url}).on(SESSION_STATUS.ESTABLISHED, (session) => {
        this.sessionStatus = SESSION_STATUS.ESTABLISHED;
        this.sessionStatusClass = 'text-success';
        this.onConnected(session);
      }).on(SESSION_STATUS.DISCONNECTED, () => {
        ...
      }).on(SESSION_STATUS.FAILED, () => {
        ...
      });
    }
    ...
}
```

4. Публикация потока

`Session.createStream()`, `Stream.publish()` code

```
publishStream() {
    let session = this.session;
    let streamName = this.publishStreamName;
    let localVideo = this.localVideo;

    if(session && localVideo) {
      session.createStream({
        name: streamName,
        display: localVideo,
        cacheLocalResources: true,
        receiveVideo: false,
        receiveAudio: false
      }).on(STREAM_STATUS.PUBLISHING, (stream) => {
        ...
      }).on(STREAM_STATUS.UNPUBLISHED, () => {
        ...
      }).on(STREAM_STATUS.FAILED, () => {
        ...
      }).publish();
    }
}
```

5. Получение события, подтверждающего публикацию

`STREAM_STATUS.PUBLISHING` code

```
publishStream() {
    let session = this.session;
    let streamName = this.publishStreamName;
```

```
    let localVideo = this.localVideo;

    if(session && localVideo) {
      session.createStream({
        ...
      }).on(STREAM_STATUS.PUBLISHING, (stream) => {
        this.publishStatus = STREAM_STATUS.PUBLISHING;
        this.publishStatusClass = 'text-success';
        this.onPublishing(stream);
      }).on(STREAM_STATUS.UNPUBLISHED, () => {
        ...
      }).on(STREAM_STATUS.FAILED, () => {
        ...
      }).publish();
    }
}
```

6. Воспроизведение потока, изменение размера отображаемой картинки

`Session.createStream()`, `Stream.play()`, `STREAM_STATUS.PENDING`, `FPUtils.resizeVideo()` code

```
playStream() {
    let session = this.session;
    let streamName = this.playStreamName;
    let remoteVideo = this.remoteVideo;

    if(session && remoteVideo) {
      session.createStream({
        name: streamName,
        display: remoteVideo
      }).on(STREAM_STATUS.PENDING, (stream) => {
        let video = document.getElementById(stream.id());
        if (!video.hasListeners) {
          video.hasListeners = true;
          video.addEventListener('resize', (event) => {
              FPUtils.resizeVideo(event.target);
          });
        }
      }).on(STREAM_STATUS.PLAYING, (stream) => {
        ...
      }).on(STREAM_STATUS.STOPPED, () => {
        ...
      }).on(STREAM_STATUS.FAILED, () => {
        ...
      }).play();
    }
}
```

7. Получение события, подтверждающего воспроизведение

`STREAM_STATUS.PLAYING` code

```
playStream() {
   let session = this.session;
   let streamName = this.playStreamName;
   let remoteVideo = this.remoteVideo;

   if(session && remoteVideo) {
     session.createStream({
       name: streamName,
       display: remoteVideo
     }).on(STREAM_STATUS.PENDING, (stream) => {
       ...
     }).on(STREAM_STATUS.PLAYING, (stream) => {
       this.playStatus = STREAM_STATUS.PLAYING;
       this.playStatusClass = 'text-success';
       this.onPlaying(stream);
     }).on(STREAM_STATUS.STOPPED, () => {
       ...
     }).on(STREAM_STATUS.FAILED, () => {
       ...
     }).play();
   }
}
```

8. Остановка воспроизведения

`Stream.stop()` code

```
onPlayClick() {
   let stream = this.playStreamObj;
   ...

   if (!stream) {
     ...
     this.playStream();
   } else {
     this.playButtonDisabled = true;
     stream.stop();
   }
}
```

9. Получение события, подтверждающего остановку воспроизведения

`STREAM_STATUS.STOPPED` code

```
playStream() {
   let session = this.session;
   let streamName = this.playStreamName;
   let remoteVideo = this.remoteVideo;

   if(session && remoteVideo) {
     session.createStream({
       name: streamName,
       display: remoteVideo
```

```
      }).on(STREAM_STATUS.PENDING, (stream) => {
        ...
      }).on(STREAM_STATUS.PLAYING, (stream) => {
        ...
      }).on(STREAM_STATUS.STOPPED, () => {
        this.playStatus = STREAM_STATUS.STOPPED;
        this.playStatusClass = 'text-success';
        this.onStopped();
      }).on(STREAM_STATUS.FAILED, () => {
        ...
      }).play();
    }
}
```

10. Остановка публикации

`Stream.stop()` code

```
onPublishClick() {
    let stream = this.publishStreamObj;
    ...

    if (!stream) {
      ...
      this.publishStream();
    } else {
      this.publishButtonDisabled = true;
      stream.stop();
    }
}
```

11. Получение события, подтверждающего остановку публикации

`STREAM_STATUS.UNPUBLISHED` code

```
publishStream() {
    let session = this.session;
    let streamName = this.publishStreamName;
    let localVideo = this.localVideo;

    if(session && localVideo) {
      session.createStream({
        ...
      }).on(STREAM_STATUS.PUBLISHING, (stream) => {
        ...
      }).on(STREAM_STATUS.UNPUBLISHED, () => {
        this.publishStatus = STREAM_STATUS.UNPUBLISHED;
        this.publishStatusClass = 'text-success';
        this.onUnpublished();
      }).on(STREAM_STATUS.FAILED, () => {
        ...
      }).publish();
```

```
    }
}
```

12. Закрытие соединения с сервером

`Session.disconnect()` code

```
onConnectClick() {
    let url = this.serverUrl;
    let session = this.session;

    if (!session) {
      ...
    } else {
      this.connectButtonDisabled = true;
      session.disconnect();
    }
}
```

13. Получение события, подтверждающего закрытие соединения

`SESSION_STATUS.DISCONNECTED` code

```
onConnectClick() {
    let url = this.serverUrl;
    let session = this.session;

    if (!session) {
      ...
      Flashphoner.createSession({urlServer:
url}).on(SESSION_STATUS.ESTABLISHED, (session) => {
        ...
      }).on(SESSION_STATUS.DISCONNECTED, () => {
        this.sessionStatus = SESSION_STATUS.DISCONNECTED;
        this.sessionStatusClass = 'text-success';
        this.onDisconnected();
      }).on(SESSION_STATUS.FAILED, () => {
        ...
      });
    }
    ...
}
```