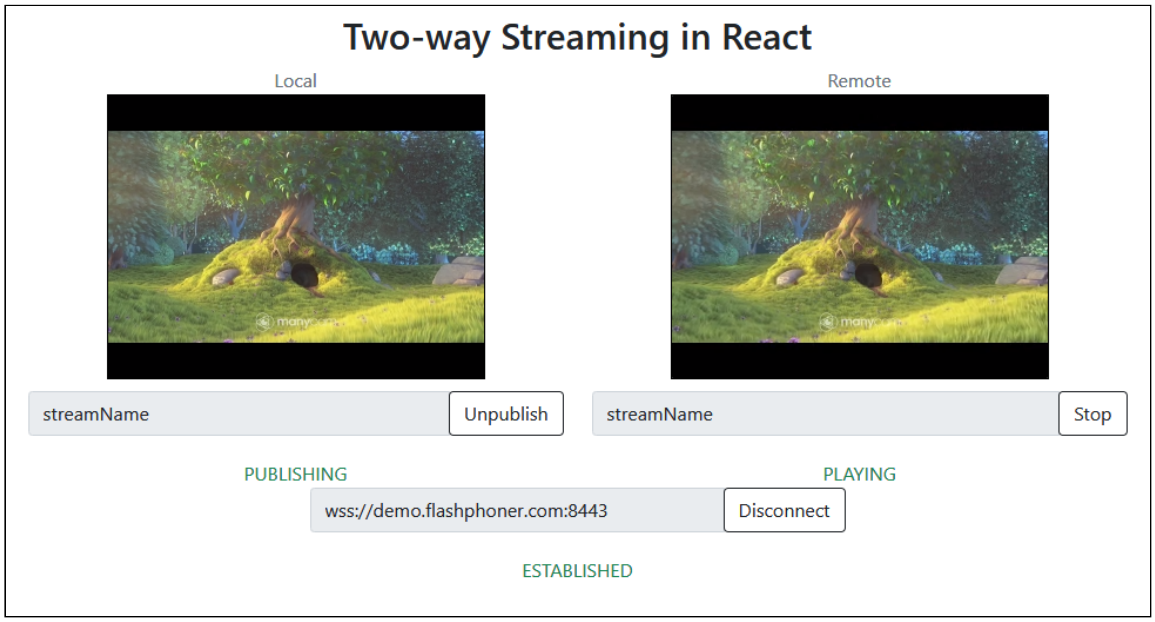
Two Way Streaming React

Описание

Приложение Two Way Streaming React показывает пример использования Web SDK в React 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-react, начиная со сборки 2.0.201.

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/react/two-way-streaming-react
```

3. Установите зависимости

```
npm install
```

4. Запустите сборку для локального тестирования

```
npm start
```

или для выгрузки на свой веб-сервер

```
npm run build
```

Работа с кодом примера

Для разбора исходного кода примера возьмем версию с хэшем `456b1c7`, которая доступна здесь

Код приложения расположен в файле TwoWayStreamingApp.js, 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

```
componentDidMount() {
 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 app = this;
```

```
  let url = this.state.serverUrl;
  let session = this.state.session;

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

4. Публикация потока

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

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

```
  let session = this.state.session;
  let streamName = this.state.publishStreamName;
  let localVideo = this.state.localVideo;

  if(session && localVideo) {
    session.createStream({
      ...
    }).on(STREAM_STATUS.PUBLISHING, (stream) => {
      app.setState({publishStatus: STREAM_STATUS.PUBLISHING,
publishStatusClass: 'text-success'});
      app.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 app = this;
  let session = this.state.session;
  let streamName = this.state.playStreamName;
  let remoteVideo = this.state.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 app = this;
 let session = this.state.session;
 let streamName = this.state.playStreamName;
 let remoteVideo = this.state.remoteVideo;

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

8. Остановка воспроизведения

`Stream.stop()` code

```
onPlayClick = () => {
 let app = this;
 let stream = this.state.playStream;
 ...

 if (!stream) {
   ...
   app.playStream();
 } else {
   app.setState({playButtonDisabled: true});
   stream.stop();
 }
}
```

9. Получение события, подтверждающего остановку воспроизведения

`STREAM_STATUS.STOPPED` code

```
playStream = () => {
 let app = this;
```

```
  let session = this.state.session;
  let streamName = this.state.playStreamName;
  let remoteVideo = this.state.remoteVideo;

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

10. Остановка публикации

`Stream.stop()` code

```
onPublishClick = () => {
  let app = this;
  let stream = this.state.publishStream;
  ...
  if (!stream) {
    ...
    app.publishStream();
  } else {
    app.setState({publishButtonDisabled: true});
    stream.stop();
  }
}
```

11. Получение события, подтверждающего остановку публикации

`STREAM_STATUS.UNPUBLISHED` code

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

  if(session && localVideo) {
    session.createStream({
      ...
    }).on(STREAM_STATUS.PUBLISHING, (stream) => {
```

```
      ...
    }).on(STREAM_STATUS.UNPUBLISHED, () => {
      app.setState({publishStatus: STREAM_STATUS.UNPUBLISHED,
publishStatusClass: 'text-success'});
      app.onUnpublished();
    }).on(STREAM_STATUS.FAILED, () => {
      ...
    }).publish();
  }
}
```

12. Закрытие соединения с сервером

`Session.disconnect()` code

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

 if (!session) {
   ...
 } else {
   app.setState({connectButtonDisabled: true});
   session.disconnect();
 }
}
```

13. Получение события, подтверждающего закрытие соединения

`SESSION_STATUS.DISCONNECTED` code

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

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