

main.js

1. Локальные переменные

Объявление локальных переменных для работы с константами, SFU SDK, для отображения локального видео и работы с конфигурацией клиента

code

```
const constants = SFU.constants;
const sfu = SFU;
let localDisplay;
let cControls;
```

2. Конфигурация по умолчанию

Объявление конфигурации комнаты и публикации потоков по умолчанию, на случай, если нет файла конфигурации `config.json`

code

Клиент настраивается на соединение с сервером по WSS по адресу localhost для входа в комнату "ROOM1" с пин-кодом "1234" под именем "Alice". Секция media задает публикацию аудио и видео дорожек. Видео публикуется двумя дорожками с качествами high (h) и medium (m)

```
const defaultConfig = {
  room: {
    url: "wss://127.0.0.1:8888",
    name: "ROOM1",
    pin: "1234",
    nickName: "Alice"
  },
  media: {
    audio: {
      tracks: [
        {
          source: "mic",
          channels: 1
        }
      ]
    },
    video: {
      tracks: [
        {
          source: "camera",
```

```

        width: 1280,
        height: 720,
        codec: "H264",
        encodings: [
            {rid: "m", active: true, maxBitrate: 300000,
scaleResolutionDownBy: 2},
            {rid: "h", active: true, maxBitrate: 900000}
        ]
    }
}
}
};

```

3. Инициализация

`init()` code

Функция `init()` вызывается после того, как страница загрузится. Функция загружает `config.json` или конфигурацию по умолчанию и открывает модальное окно входа

```

/**
 * load config and show entrance modal
 */
const init = function () {
    //read config
    $.getJSON("config.json", function (config) {
        cControls = createControls(config);
    }).fail(function () {
        //use default config
        cControls = createControls(defaultConfig);
    });
    //open entrance modal
    $('#entranceModal').modal('show');
}

```

4. Соединение с сервером и создание либо вход в комнату

`connect()` code

Функция вызывается по щелчку пользователя по кнопке Enter в модальном окне входа

```

/**
 * connect to server
 */
async function connect() {
    // hide modal
    $('#entranceModal').modal('hide');
}

```

```

// disable controls
cControls.muteInput();
//create peer connection
const pc = new RTCPeerConnection();
//get config object for room creation
const roomConfig = cControls.roomConfig();
//kick off connect to server and local room creation
try {
  const session = await sfu.createRoom(roomConfig);
  // Now we connected to the server (if no exception was thrown)
  session.on(constants.SFU_EVENT.FAILED, function (e) {
    if (e.status && e.statusText) {
      displayError("CONNECTION FAILED: " + e.status + " " +
e.statusText);
    } else if (e.type && e.info) {
      displayError("CONNECTION FAILED: " + e.info);
    } else {
      displayError("CONNECTION FAILED: " + e);
    }
  }).on(constants.SFU_EVENT.DISCONNECTED, function (e) {
    displayError("DISCONNECTED. Refresh the page to enter the room
again");
  });
  const room = session.room();
  room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {
    displayError(e);
  }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
    displayError(e.operation + " failed: " + e.error);
  })

  // create local display to show local streams
  localDisplay =
initLocalDisplay(document.getElementById("localDisplay"));
  // display audio and video control tables
  await cControls.displayTables();
  cControls.onTrack(async function (s) {
    await publishNewTrack(room, pc, s);
  });
  //create and bind chat to the new room
  const chatDiv = document.getElementById('messages');
  const chatInput = document.getElementById('localMessage');
  const chatButton = document.getElementById('sendMessage');
  createChat(room, chatDiv, chatInput, chatButton);

  //setup remote display for showing remote audio/video tracks
  const remoteDisplay = document.getElementById("display");
  initDefaultRemoteDisplay(room, remoteDisplay, {quality: true},
{thresholds: [
    {parameter: "nackCount", maxLeap: 10},
    {parameter: "freezeCount", maxLeap: 10},
    {parameter: "packetsLost", maxLeap: 10}
  ], abrKeepOnGoodQuality: ABR_KEEP_ON_QUALITY,
abrTryForUpperQuality: ABR_TRY_UPPER_QUALITY, interval:
ABR_QUALITY_CHECK_PERIOD});

  //get configured local video streams
  let streams = cControls.getVideoStreams();
  //combine local video streams with audio streams

```

```

streams.push.apply(streams, cControls.getAudioStreams());

// Publish preconfigured streams
publishPreconfiguredStreams(room, pc, streams);
} catch (e) {
  console.error(e);
  displayError(e);
}
}
}

```

5. Подробнее о функции connect()

Скрытие модального окна входа и отключение полей ввода до установки соединения с сервером

code

```

async function connect() {
  // hide modal
  $('#entranceModal').modal('hide');
  // disable controls
  cControls.muteInput();
  ...
}

```

Создание объекта PeerConnection и подготовка объекта конфигурации комнаты

code

```

async function connect() {
  ...
  //create peer connection
  const pc = new RTCPeerConnection();
  //get config object for room creation
  const roomConfig = cControls.roomConfig();
  ...
}

```

Создание сессии и установка соединения с сервером

code

```

async function connect() {
  ...
  //kick off connect to server and local room creation
  try {
    const session = await sfu.createRoom(roomConfig);
    ...
  } catch (e) {
    console.error(e);
    displayError(e);
  }
}

```

```
}  
}
```

Подписка на события сессии

code

```
async function connect() {  
  ...  
  //kick off connect to server and local room creation  
  try {  
    ...  
    // Now we connected to the server (if no exception was thrown)  
    session.on(constants.SFU_EVENT.FAILED, function (e) {  
      if (e.status && e.statusText) {  
        displayError("CONNECTION FAILED: " + e.status + " " +  
e.statusText);  
      } else if (e.type && e.info) {  
        displayError("CONNECTION FAILED: " + e.info);  
      } else {  
        displayError("CONNECTION FAILED: " + e);  
      }  
    }).on(constants.SFU_EVENT.DISCONNECTED, function (e) {  
      displayError("DISCONNECTED. Refresh the page to enter the room  
again");  
    });  
    ...  
  } catch (e) {  
    console.error(e);  
    displayError(e);  
  }  
}
```

Создание объекта комнаты и подписка на сообщения об ошибках

code

```
async function connect() {  
  ...  
  //kick off connect to server and local room creation  
  try {  
    ...  
    const room = session.room();  
    room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {  
      displayError(e);  
    }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {  
      displayError(e.operation + " failed: " + e.error);  
    })  
    ...  
  } catch (e) {  
    console.error(e);  
    displayError(e);  
  }  
}
```

Создание объекта для отображения локального видео

code

```
async function connect() {
  ...
  //kick off connect to server and local room creation
  try {
    ...
    // create local display to show local streams
    localDisplay =
    initLocalDisplay(document.getElementById("localDisplay"));
    // display audio and video control tables
    await cControls.displayTables();
    cControls.onTrack(async function (s) {
      await publishNewTrack(room, pc, s);
    });
    ...
  } catch (e) {
    console.error(e);
    displayError(e);
  }
}
```

Инициализация окна чата

code

```
async function connect() {
  ...
  //kick off connect to server and local room creation
  try {
    ...
    //create and bind chat to the new room
    const chatDiv = document.getElementById('messages');
    const chatInput = document.getElementById('localMessage');
    const chatButton = document.getElementById('sendMessage');
    createChat(room, chatDiv, chatInput, chatButton);
    ...
  } catch (e) {
    console.error(e);
    displayError(e);
  }
}
```

Инициализация объекта для отображения потоков от других участников

code

```
async function connect() {
  ...
  //kick off connect to server and local room creation
  try {
    ...
```

```

//setup remote display for showing remote audio/video tracks
const remoteDisplay = document.getElementById("display");
initDefaultRemoteDisplay(room, remoteDisplay, {quality: true},
{thresholds: [
  {parameter: "nackCount", maxLeap: 10},
  {parameter: "freezeCount", maxLeap: 10},
  {parameter: "packetsLost", maxLeap: 10}
], abrKeepOnGoodQuality: ABR_KEEP_ON_QUALITY,
abrTryForUpperQuality: ABR_TRY_UPPER_QUALITY, interval:
ABR_QUALITY_CHECK_PERIOD});
...
} catch (e) {
  console.error(e);
  displayError(e);
}
}

```

Получение настроек и публикация локальных медиа потоков

code

```

async function connect() {
  ...
  //kick off connect to server and local room creation
  try {
    ...
    //get configured local video streams
    let streams = cControls.getVideoStreams();
    //combine local video streams with audio streams
    streams.push.apply(streams, cControls.getAudioStreams());

    // Publish preconfigured streams
    publishPreconfiguredStreams(room, pc, streams);
  } catch (e) {
    console.error(e);
    displayError(e);
  }
}

```

6. Вход в комнату и публикация локальных потоков из файла конфигурации

`publishPreconfiguredStreams()`, `Room.join()` code

```

const publishPreconfiguredStreams = async function (room, pc, streams) {
  try {
    const config = {};
    //add our local streams to the room (to PeerConnection)
    streams.forEach(function (s) {
      let contentType = s.type || s.source;
      //add each track to PeerConnection
      s.stream.getTracks().forEach((track) => {
        config[track.id] = contentType;
      });
    });
  }
}

```

```

        addTrackToPeerConnection(pc, s.stream, track, s.encodings);
        subscribeTrackToEndedEvent(room, track, pc);
    });
    localDisplay.add(s.stream.id, "local", s.stream, contentType);
});
//join room
await room.join(pc, null, config, 10);
// Enable Delete button for each preconfigured stream #WCS-3689
streams.forEach(function (s) {
    $('#' + s.stream.id + "-button").prop('disabled', false);
});
} catch (e) {
    onOperationFailed("Failed to publish a preconfigured streams", e);
    // Enable Delete button for each preconfigured stream #WCS-3689
    streams.forEach(function (s) {
        $('#' + s.stream.id + "-button").prop('disabled', false);
    });
}
}
}

```

7. Публикация дополнительных локальных потоков

`publishNewTrack()`, `Room.updateState()` [code](#)

```

const publishNewTrack = async function (room, pc, media) {
    try {
        let config = {};
        //add local stream to local display
        let contentType = media.type || media.source;

        localDisplay.add(media.stream.id, "local", media.stream,
contentType);
        //add each track to PeerConnection
        media.stream.getTracks().forEach((track) => {
            config[track.id] = contentType;
            addTrackToPeerConnection(pc, media.stream, track,
media.encodings);
            subscribeTrackToEndedEvent(room, track, pc);
        });
        // Clean error message
        displayError("");
        //kickoff renegotiation
        await room.updateState(config);
        // Enable Delete button for a new stream #WCS-3689
        $('#' + media.stream.id + "-button").prop('disabled', false);
    } catch (e) {
        onOperationFailed("Failed to publish a new track", e);
        // Enable Delete button for a new stream #WCS-3689
        $('#' + media.stream.id + "-button").prop('disabled', false);
    }
}
}

```


8. Завершение публикации потока

`subscribeTrackToEndedEvent()` [code](#)

Вспомогательная функция, которая подписывается на событие "ended" для локального потока. При получении события поток удаляется из PeerConnection, и состояние комнаты обновляется

```
const subscribeTrackToEndedEvent = function (room, track, pc) {
  track.addEventListener("ended", async function () {
    try {
      //track ended, see if we need to cleanup
      let negotiate = false;
      for (const sender of pc.getSenders()) {
        if (sender.track === track) {
          pc.removeTrack(sender);
          //track found, set renegotiation flag
          negotiate = true;
          break;
        }
      }
      // Clean error message
      displayError("");
      if (negotiate) {
        //kickoff renegotiation
        await room.updateState();
      }
    } catch (e) {
      onOperationFailed("Failed to update room state", e);
    }
  });
}
```

9. Добавление новой дорожки в PeerConnection

`addTrackToPeerConnection()` [code](#)

Вспомогательная функция, которая добавляет новую дорожку в PeerConnection для публикации

```
const addTrackToPeerConnection = function(pc, stream, track, encodings) {
  pc.addTransceiver(track, {
    direction: "sendonly",
    streams: [stream],
    sendEncodings: encodings ? encodings : [] //passing encoding types
  });
}
```