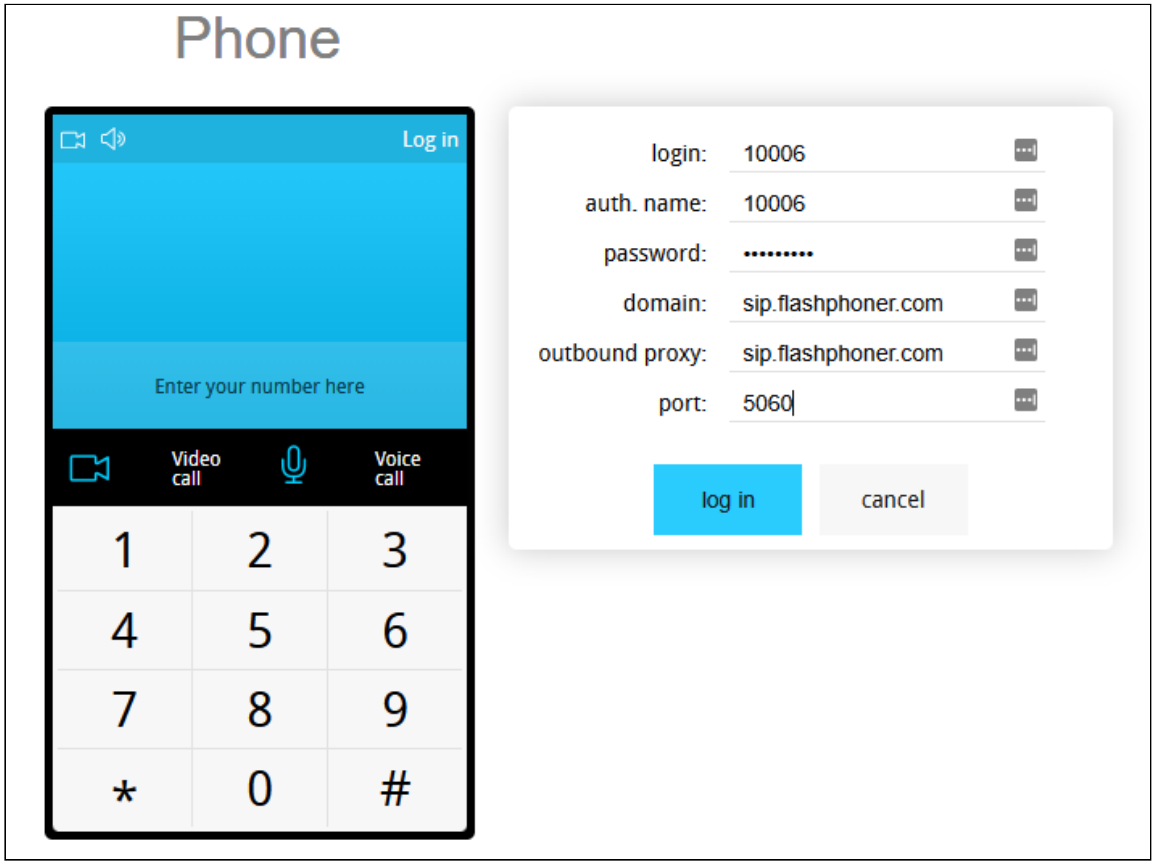
Phone UI

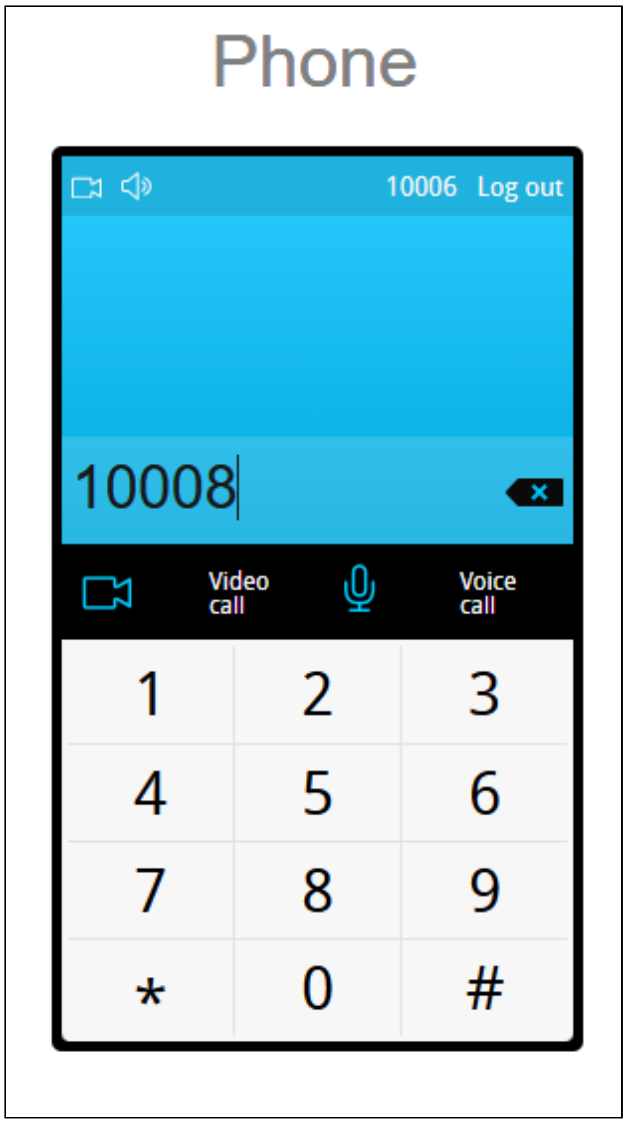
Example of web phone with dialer GUI

On the screenshots below

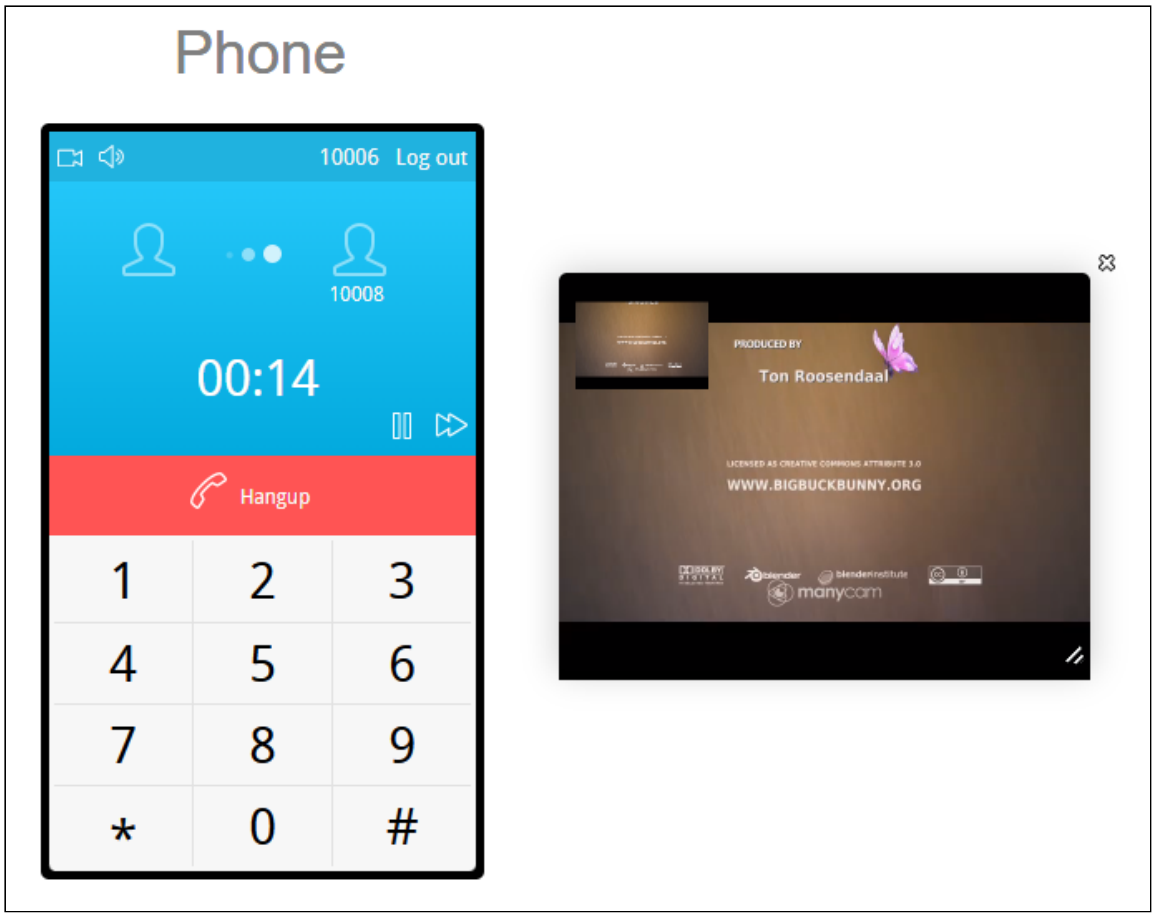
- parameters for SIP connection are entered in to the login form



- callee SIP username is entered



- video call is established



Video or voice call is establishing when corresponding button is clicked and is terminated when Hangup button is clicked. Call can be put on hold using the `pause` ⏸ icon. The volume can be adjusted using the `loudspeaker` 🔊 icon in the top left corner.

Code of the example

The path to the source code of the example on WCS server is:

/usr/local/FlashphonerWebCallServer/client/examples/demo/sip/phone-ui

- gui - directory containing files required for the GUI: file with styles, fonts, ../images
- listener - directory containing event listener scripts
- sounds - directory containing sound files for events
- SoundControl.js - script providing functionality for playing sounds
- Phone.js - script providing functionality for the web phone
- Phone.html - page of the web phone

This example can be tested using the following address:

https://host:8888/client/examples/demo/sip/phone-ui/Phone.html

Here host is the address of the WCS server.

Analyzing the code

To analyze the code, let's take the version of file `Phone.js` with hash `ecbadc3`, which is available here and can be downloaded with corresponding build 2.0.212.

In this script, Flashphoner API methods are called from the corresponding methods of Phone object. For example, method `createSession()` for establishing connection to server is called from method `connect()`

code

```
Phone.prototype.connect = function () {
    var me = this;
    ...

Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session){
    ...
}
```

The Phone object is created and initialized after API initialization

code

```
var phone = new Phone();

phone.init();
```

In addition to the methods required for the calling functionality, Phone object has listener methods for changing controls of the interface depending on the connection and call status (line 194 - line 416).

1. Initialization of the API

Flashphoner.init() code

```
Flashphoner.init();
```

2. Connection to server

`Flashphoner.createSession()` code

Object with connection options is passed to the method

- `urlServer` - URL for WebSocket connection to WCS server

- `sipOptions` - object with parameters for SIP connection

```
var url = setURL();

this.sipOptions = {};
this.sipOptions.login = $('#sipLogin').val();
this.sipOptions.password = $('#sipPassword').val();
this.sipOptions.authenticationName = $('#sipAuthenticationName').val();
this.sipOptions.domain = $('#sipDomain').val();
this.sipOptions.outboundProxy = $('#sipOutboundProxy').val();
this.sipOptions.port = $('#sipPort').val();
this.sipOptions.useProxy = true;
this.sipOptions.registerRequired = true;

var connectionOptions = {
    urlServer: url,
    sipOptions: this.sipOptions
};

Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session){
    ...
});
```

3. Receiving the event confirming successful connection

`ConnectionStatusEvent.ESTABLISHED` code

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session){
    me.session = session;
    me.connectionStatusListener(SESSION_STATUS.ESTABLISHED);
}).on(SESSION_STATUS.REGISTERED, function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    ...
});
```

4. Receiving the event confirming successful registration on SIP server

`ConnectionStatusEvent.REGISTERED` code

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session){
    ...
}).on(SESSION_STATUS.REGISTERED, function(session){
    me.registrationStatusListener(SESSION_STATUS.REGISTERED);
}).on(SESSION_STATUS.DISCONNECTED, function(){
```

```
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    ...
});
```

5. Receiving the event on incoming call

`ConnectionStatusEvent.INCOMING_CALL` code

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session){
    ...
}).on(SESSION_STATUS.REGISTERED, function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    call.on(CALL_STATUS.RING, function(){
        ...
    });
    me.onCallListener(call);
});
```

6. Outgoing call

`Session.createCall()`, `Call.call()` code

The following parameters are passed when call is created

- `callee` - callee SIP username
- `visibleName` - display name
- `localVideoDisplay` - `div` element to display local video/audio
- `remoteVideoDisplay` - `div` element to display remote video/audio
- `constraints` - object with parameters specifying if the call should have audio and video

```
var constraints = {
    audio: true,
    video: hasVideo
};

var outCall = this.session.createCall({
    callee: callee,
    visibleName: this.sipOptions.login,
    localVideoDisplay: this.localVideo,
    remoteVideoDisplay: this.remoteVideo,
    constraints: constraints
```

```
    ...
});

outCall.call();
```

7. Answering incoming call

`Call.answer()` code

Object with answer options is passed to the method

- `localVideoDisplay` - `div` element to display local video/audio
- `remoteVideoDisplay` - `div` element to display remote video/audio

```
Phone.prototype.answer = function () {
    trace("Phone - answer " + this.currentCall.id());
    this.flashphonerListener.onAnswer(this.currentCall.id());
    this.currentCall.answer({
        localVideoDisplay: this.localVideo,
        remoteVideoDisplay: this.remoteVideo
    });
};
```

8. Call hold

- put on hold: `Call.hold()` code

  ```
  Phone.prototype.hold = function () {
      trace("Phone - hold callId: " + this.currentCall.id());
      this.currentCall.hold();
  };
  ```

- retrieve: `Call.unhold()` code

  ```
  Phone.prototype.unhold = function () {
      trace("Phone - hold callId: " + this.currentCall.id());
      this.currentCall.unhold();
  };
  ```

9. Call hangup

`Call.hangup()` code

```
Phone.prototype.hangup = function () {
    trace("Phone - hangup " + this.currentCall.id() + " status " +
this.currentCall.status());
    this.hideFlashAccess();
```

```
    if (this.currentCall.status() == CALL_STATUS.PENDING) {
        this.callStatusListener(this.currentCall);
    } else {
        this.currentCall.hangup();
    }
    this.flashphonerListener.onHangup();
};
```

10. Disconnection

`Session.disconnect()` code

```
Phone.prototype.disconnect = function () {
    trace("Phone - disconnect");
    this.session.disconnect();
};
```