

Web Integration Scenarios

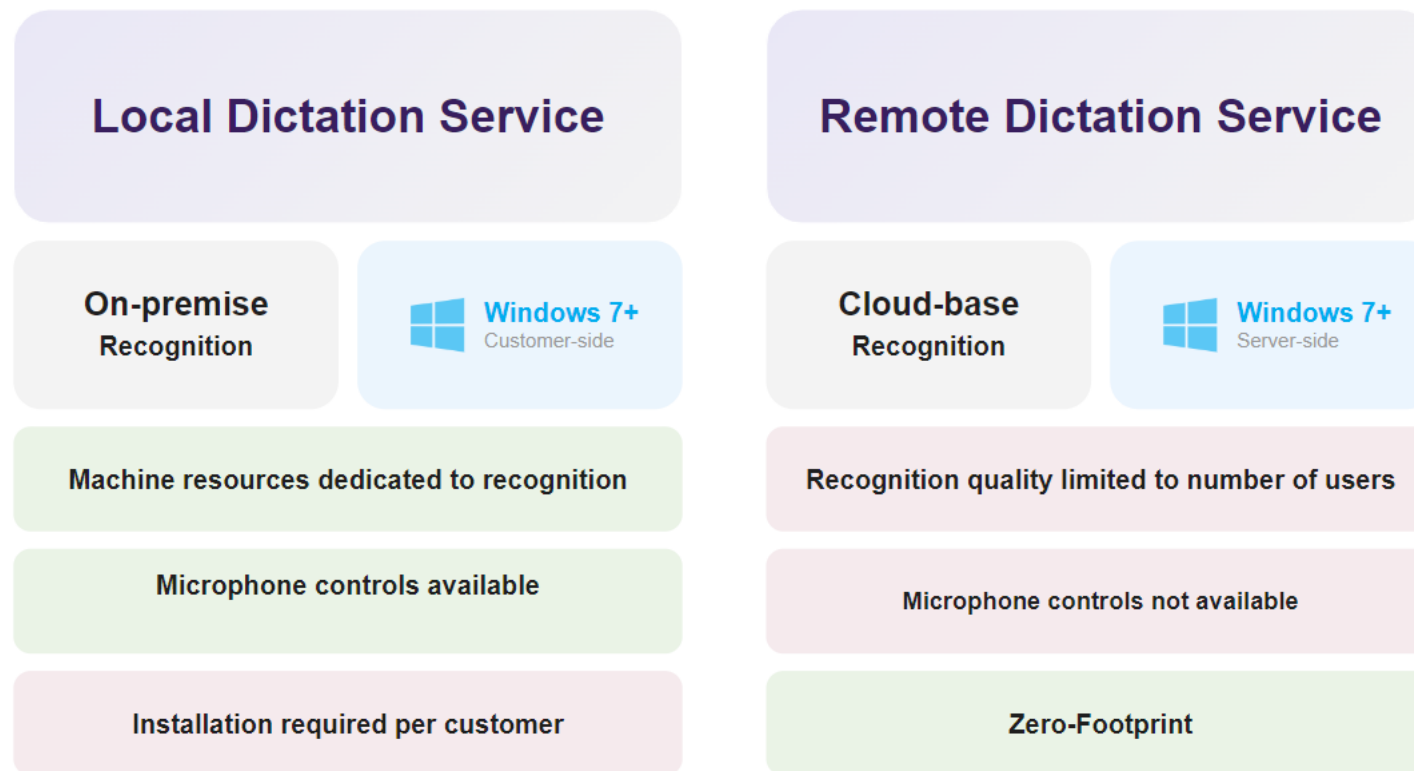


Figure 1: Scenarios

Two web integration scenarios are available based on your requirements:

- **Local Dictation Service:** This scenario offers the installation and operation of the recognition service within the client's local environment. This option leverages the dedicated resources of the machine, resulting in potentially higher recognition quality. Microphone controls are available, enabling greater control during the dictation process. Each client can perform the installation according to their specific needs, customizing the service to their environment and requirements. However, it is important to note that this option requires installation and configuration on each client and is limited to Windows 7 or higher operating systems.
- **Remote Dictation Service:** This scenario provides an alternative approach by utilizing cloud-based recognition, offering enhanced flexibility and multiplatform accessibility. With this service, there is no need for individual client installations as it operates from a dedicated server. However, it is important to note that

this option has limitations in terms of accommodating an unlimited number of users, as it is subject to the computational resources of the server. Furthermore, the quality of recognition may be impacted based on the concurrent user load. It is worth mentioning that microphone controls are not available within this option, thereby restricting direct control during the dictation process.

Both scenarios can be deployed simultaneously within the same integration if needed. This means that if required, you can utilize both the Local Dictation Service and the Remote Dictation Service concurrently. This provides flexibility and allows you to tailor the integration to your specific requirements.

Local Dictation Service

Description

The integration of **INVOX Medical with Local Dictation Service** is suitable for customers with Windows 7 operating systems or higher.

It provides the necessary tools to integrate INVOX Medical speech recognition and all its functionalities in a web environment.

The API provides from the most basic and essential functions to be able to log in and dictate in simple steps, to advanced functions to control dictation, obtain session information or even customize the behaviour of INVOX Medical components. All the low-level details related to speech recognition are hidden from the integrator.

On the other hand, **the Local Dictation Service** will contain the INVOX Medical voice recognition engine, which **must be installed on the customer's equipment**.

Integration architecture

Integration into the external application

To include INVOX Medical in your web application, simply include the library provided, import it into your application, and execute the essential functions discussed in the API.

The web integration mechanism with Local Dictation Service requires the installation of the INVOX Medical Local Dictation Service.

Steps to perform

1. **Import the INVOX Medical library into your web application.**

The `libs/invox.min.js` file contains all the necessary functionality for the correct deployment of INVOX Medical. The following HTML block shows how to import it:

```
<head>
...

<script type="text/javascript" src="libs/invox.min.js" charset="UTF-8"></script>
</head>
```

The Opus library `libs/opus/` is not used by the Local Dictation Service.

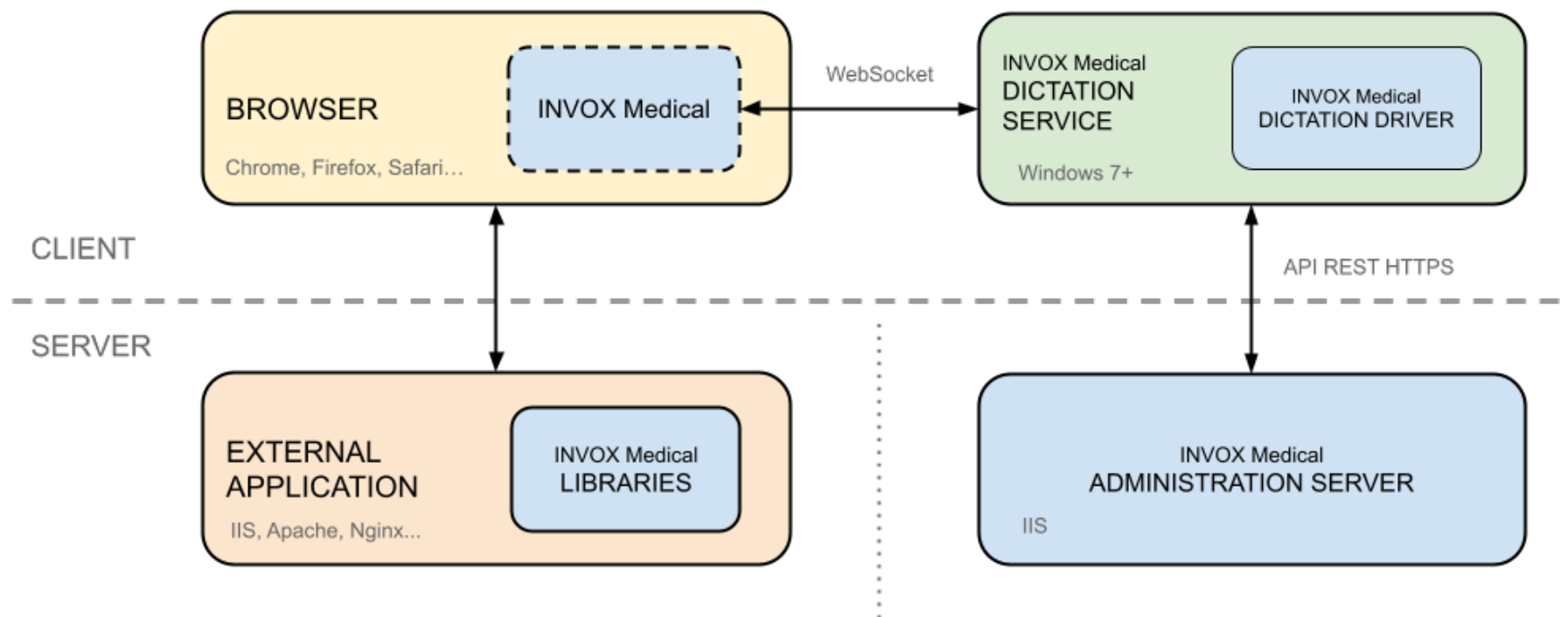


Figure 2: Integration architecture

1. Install the Local Dictation Service.

Ensure that the Local Dictation Service has been installed and is functioning correctly.

2. Start integrating INVOX Medical into your application.

You can develop your own implementation or you can use INVOX Medical integrations to facilitate and accelerate deployment.

For the first case, you can start with the quick start guide where we explain the minimum components you need to get started. You can also consult the complete list of the functions that make up the INVOX Medical API.

On the other hand, we have prepared a section explaining the integration examples, accessible from the `libs/samples` folder. In this section you can understand how to implement the different integrations in your web application.

Remarks

INVOX Medical will attempt to connect via the **WebSocket** protocol using the browser's native methods. It will be necessary to install the certificate of the dictation service in order to be able to use the audio sending functionality.

Deployment on clients

If you have opted for the Local Dictation Service scenario for your integration, you will need to request the installer from the <https://support.invoxmedical.com/>. Visit the link provided to submit your request and get the installers required for your deployment.

1. Install the **Dictation Drivers** in the customer's operating system.

`InvoxMD_DictationDriver_DD_Install_X.X.X_x64.exe` (64 bits).

2. Install the provided **Local Dictation Service** on the customer's operating system:

`InvoxMDAgent_XXX_CL_Install_X.X.X_x64.exe` (64 bits).

You can also find the Dictation Drivers at <https://sdk.invoxmedical.com/sdkportal28/Downloads>.

Logging

See the Logging section for details of the Local Dictation Service Log.

Remote Dictation Service

Description

The Remote Dictation Service integration of INVOX Medical is **suitable for clients with non-Windows operating systems, such as Linux or macOS**, and has certain limitations in the control of audio and microphones compared to other integration methods.

It provides the tools necessary to integrate INVOX Medical speech recognition and all its functionalities into a web environment.

The API provides from the most basic and essential functions to be able to log in and dictate in simple steps, to advanced functions to control dictation, obtain session information or even customise the behaviour of INVOX Medical components. Hiding all the low-level details related to speech recognition from the integrator.

The INVOX Medical speech recognition engine is deployed on a server, so no additional installations are required on the customer's equipment.

Integration architecture

Integration into the external application

To include INVOX Medical in your web application, simply include the library provided, import it into your application, and execute the essential functions discussed in the API.

The Remote Dictation Service web integration mechanism also requires ensuring that the `libs/opus` folder is included in the directory of your web application. This library will take care of the audio management.

Steps to perform

1. Import the INVOX Medical library into your web application.

The `libs/invox.min.js` file contains all the necessary functionality for the correct deployment of INVOX Medical. The following HTML block shows how to import it:

```
<head>
...

<script type="text/javascript" src="libs/invox.min.js" charset="UTF-8"></script>
</head>
```

The Opus library `libs/opus/` is mandatory for the Remote Dictation Service.

INVOX Medical will automatically reference the `libs/opus/` directory, so it should not be imported into your web application. But it is important to keep the same folder structure:

```
| -invox.min.js
| -opus/
```

1. Enable mimetype wasm in dictation server

This mimetype is necessary to send audio from browser with HTML5.

- Open **Internet Information Services Manager (IIS)**. You can find it by typing “IIS” in the search field.

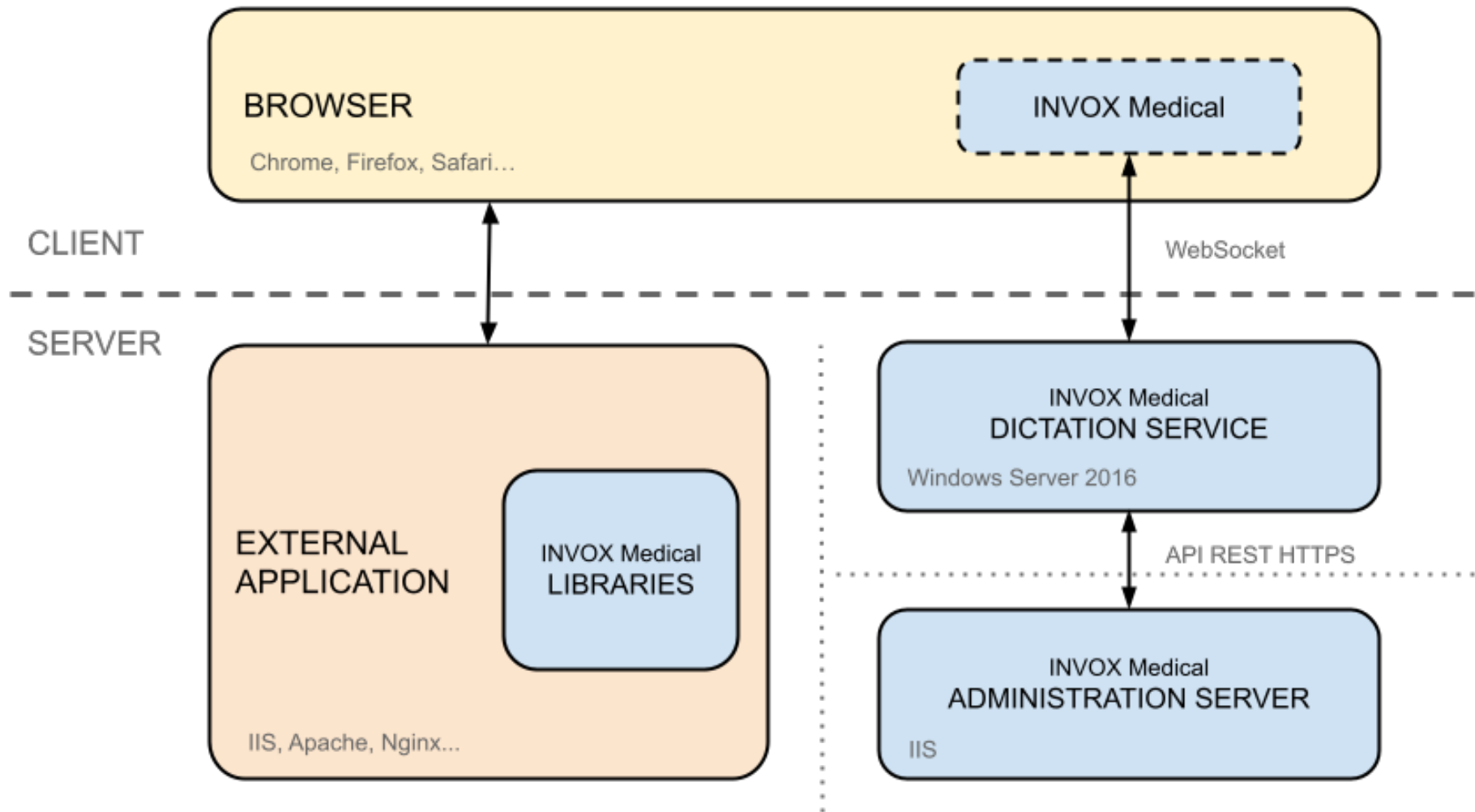


Figure 3: Integration architecture

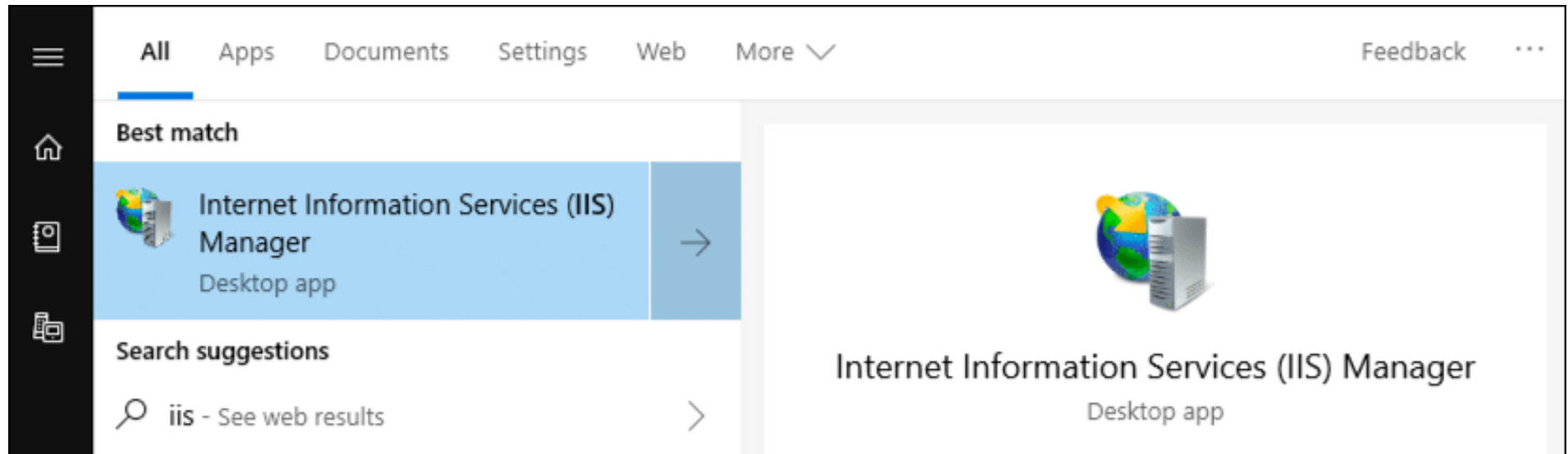
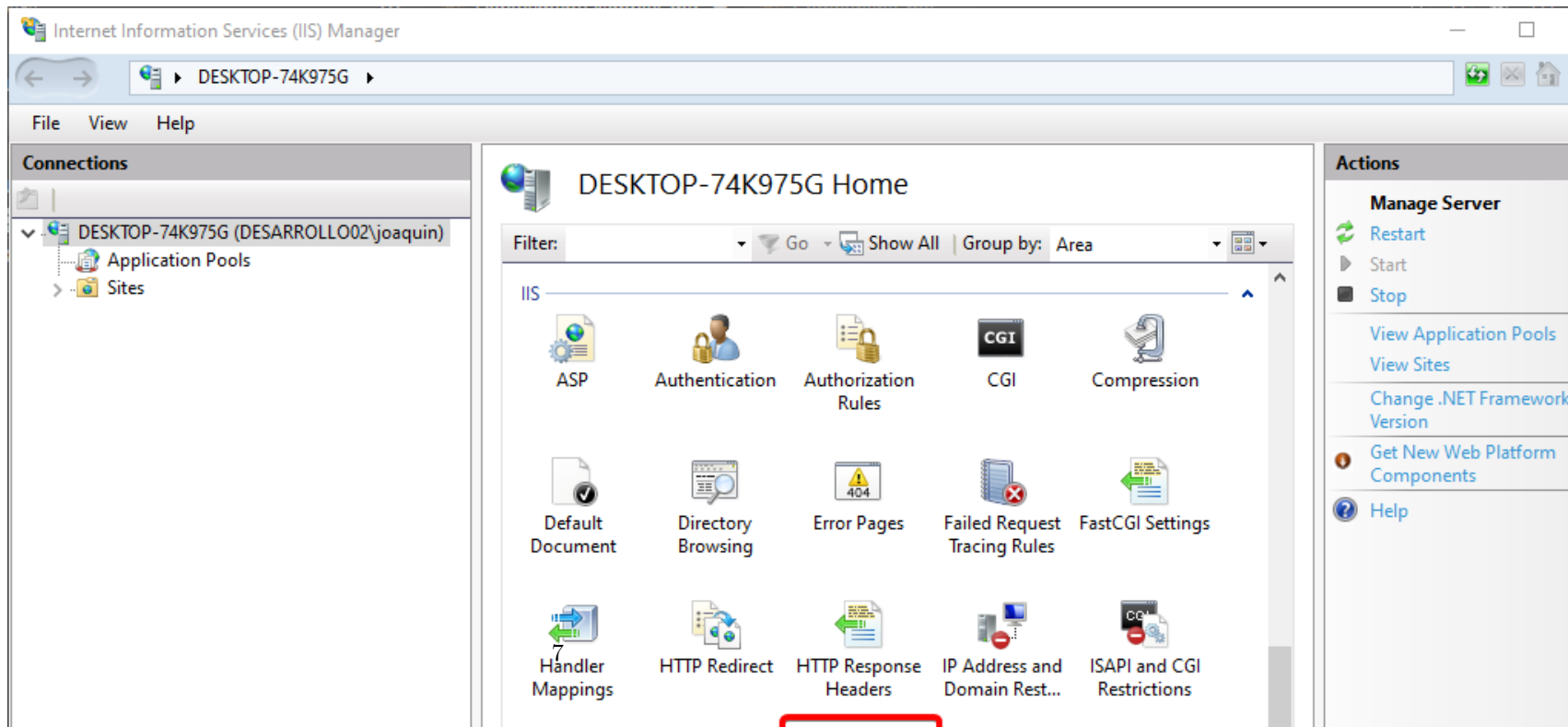


Figure 4: Search



- Right click, select *Add...* and set the values:

Field	Value
File name extension	<i>.wasm</i>
MIME type	<i>application/wasm</i>

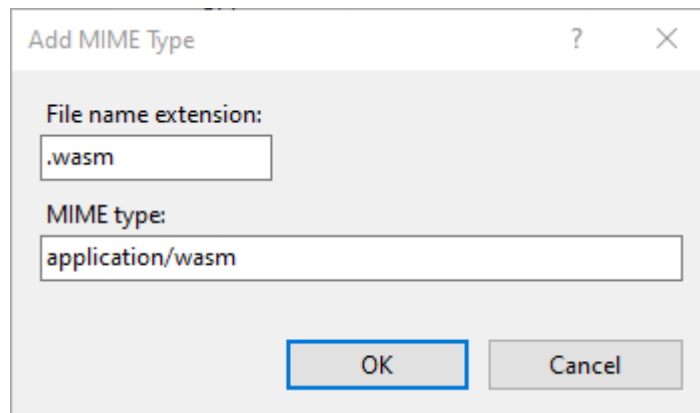


Figure 5: Search

2. Start integrating INVOX Medical into your application.

You can start with the quick start guide, and for more detail we have prepared a section explaining the integration examples, accessible from the `lib/samples` folder.

You can also consult the complete list of the functions that make up the INVOX Medical API.

Remarks

INVOX Medical will attempt to connect via the **WebSocket** protocol using the browser's native methods. It will be necessary to install the certificate of the dictation service in order to be able to use the audio sending functionality.

Deployment on clients

No installation on clients is required to deploy the integration. The client web application connects to the Remote Dictation Service via a *WebSocket* connection.

If this service does not include a valid certificate, it is necessary to include the connection exception on each client machine or to include a valid certificate to the service.

Logging

See the Logging section for details of the Remote Dictation Service Log.

Test Scenarios

Where can I find these examples?

These examples can be found at <https://sdk.invoxmedical.com/sdk28>.

In addition, the full code of these examples is provided in the `/samples` folder of the SDK release ZIP that you can download at <https://sdk.invoxmedical.com/sdkportal28/Downloads>.

What credentials do I use in these examples?

Our support team will provide you with two usernames, each with its respective password. Please use this login information to perform testing in the environment, explore its functionality, and carry out your integration.

If you have any questions or need assistance, please don't hesitate to consult with our support team at <https://support.invoxmedical.com/>.

What should I install?

No installation is required to test the Remote Dictation Service scenario as it operates from a dedicated server. However, in order to test the Local Dictation Service, installation is required on your machine. You will need to download the service and microphone driver installers from the following link: <https://sdk.invoxmedical.com/sdkportal28/Downloads>.

Our support team will provide you with a username to access the SDK Portal and thus be able to consult the documentation or download the trial installers.

IMPORTANT: The credentials for the SDK Portal and the usernames and passwords provided to test the examples serve different purposes and should not be used interchangeably.

What is the purpose of these examples?

These example pages are intended to serve as a guide for your development. They act as a **Playground** to understand some concepts about the INVOX Medical SDK.

Here you can test the connection to the different dictation services, test the available functionalities and, of course, access the code to use it in your implementations.

What can I learn from them?

In the Test Simple Components Sample you can learn:

- How to use the INVOX Medical SDK.
- How to create your own components, for example:
 - Start or to stop the recognizer.



INVOX Medical Bar & Components

INVOX Medical Bar Integration with multiple text editors

On this page you will find the INVOX Medical Bar, which is an example of integration with all the components. You can also see how it works with multiple writing destinations and how to change the focus between them.

[Open example](#) →

INVOX Medical Components

On this page you will find all the INVOX Medical components. You can see how they work and play with them.

[Open example](#) →

Integrations with CKEditor 4

INVOX Medical Bar Integration with CKEditor 4

On this page you can see how to integrate CKEditor 4 as an alternative text editor and how to change the focus between each write destination.

[Open example](#) →

INVOX Medical Components into CKEditor 4

On this page you will find how to integrate all the INVOX Medical components into CKEditor 4.

[Open example](#) →

INVOX Medical Components into CKEditor 4 with another text editor

On this page you will find all the INVOX Medical components integrated into CKEditor 4 and how to switch the focus between each write destination.

[Open example](#) →

Figure 6: Home page from INVOX Medical SDK



Installers	Description	Version	Assembly	Size
 Local Dictation Service	Local Dictation Service installer for INVOX Medical SDK	2.8.1	x64	88.07 MB
 Dictation Driver	Microphone drivers installer for Local Dictation Service ✓ <i>SpeechMike drivers</i> ✓ <i>Olympus drivers</i> ✓ <i>Nuance drivers</i>	2.8.1	x64	124.79 MB

Figure 7: Local Dictation Service Installers

- Display user session information.
- Show audio feedback.
- Etc.
- How to write in a HTML TextArea.

In the Test Quick Integration Samples you can learn:

- How to implement Quick Integrations:
 - Dictation Bar Component.
 - Dictionary Component.
 - Templates Component.
 - Transformations Component.

Test Quick Integrations

Introduction

Another possibility offered by INVOX Medical SDK is to use the Quick Integrations. Also, you can understand how the editor to use is defined, for example CKEditor 4.

All of these complex components are implemented using the INVOX Medical API functions.

Integrate Components

Check how to include these components in your web application:

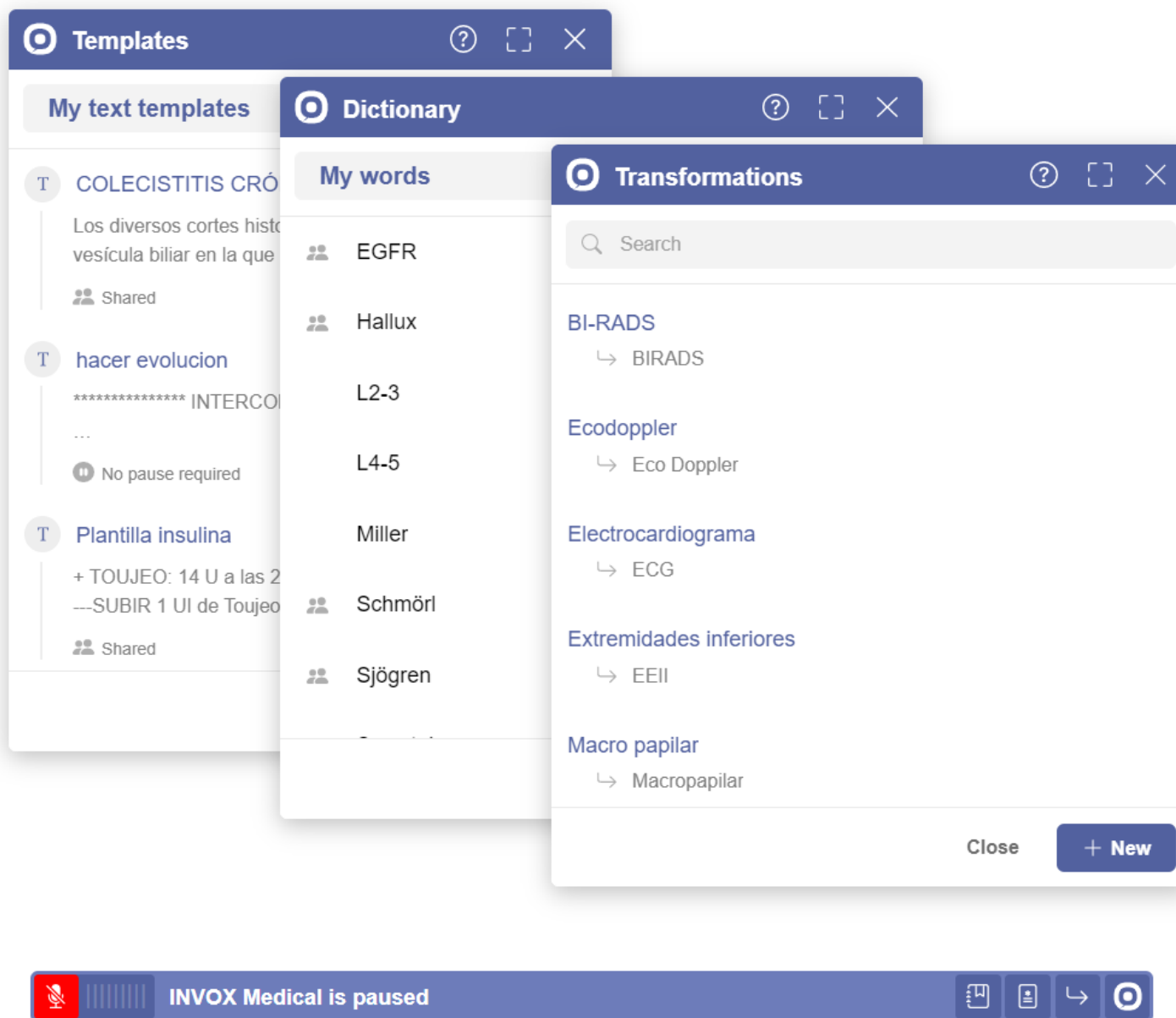


Figure 8: INVOX Medical Bar Integration with CKEditor 4

- Dictation Bar Component
- Dictionary Component
- Templates Component
- Transformations Component

Set writing location

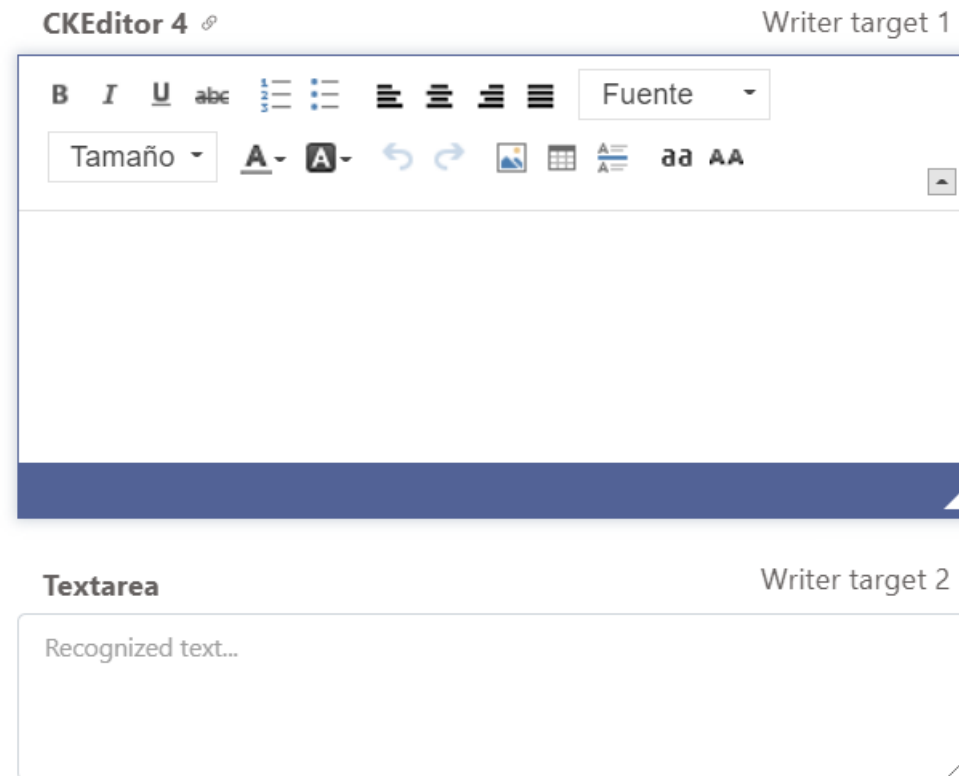


Figure 9: Setting two writing locations using TextAreas

We must specify to INVOX Medical which is the editor to be able to view, edit or move through the recognized text. In this example we use two different editors, so it is essential to have the TextWriter of each editor implemented.

INVOX Medical SDK provides the implementation of the `TextWriter` for `TextAreas` .

TextArea

The `TextArea` shall be assigned “`invox-textarea`” as a unique identifier. The HTML code to create it is as follows:

```
<div class="row invox-nomargin">
  ...

  <textarea id="invox-textarea" class="form-control invox-textarea shadow-none" placeholder="Recognized text..." onFocus="OnClickTextArea(id)"></div>
```

Each time a text box is clicked, it will be set as the current `WriterTarget`. To do this, it will execute the **`OnClickTextArea`** function, which receives the `TextArea` identifier and passes it to **`ChangeFocusToTextArea`** to change `WriterTarget`.

```
let OnClickTextArea = function (id) {
  //...
  ChangeFocusToTextArea(id);
}

let ChangeFocusToTextArea = function(id) {
  try {
    INVOX.SetTextWriter(INVOX.TextAreaTextWriter);
    INVOX.SetWriterTarget(id);
    //...
  } catch(e) {
    ShowMessage(INVOX.MessageType.ERROR, e);
  }
}
```

Create Writer controllers (optional)

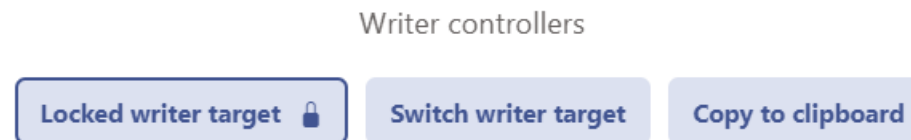


Figure 10: Buttons to control the writing focus

The Writer controllers implement very simple functions that allow you to easily manage the current WriterTarget.

- **Locked writer target.** With the button active the text recognised by INVOX Medical will be written in the current Writer Target (text box with the focus), even if you manually switch the focus to another editor.
- **Switch writer target.** Switches the focus between the WriterTargets set.
- **Copy to clipboard.** Gets the text from the current Writer Target (textbox with the focus) and copies it to the clipboard.

The HTML code to create the buttons is as follows:

```
<div class="d-flex flex-wrap justify-content-center">
  <button id="inbox-switch-lock-btn" class="btn inbox-base-btn inbox-switch-lock-btn inbox-switch-lock-focus-btn" onclick="SwitchLockTargetByFocus">
    <span id="inbox-switch-lock-btn-text">Locked writer target</span>
    <i id="inbox-lock-icon" style="margin-left: 5px; font-size: 1em;" class="bi bi-lock-fill"></i>
  </button>
  <button id="inbox-clipboard-btn" class="btn inbox-base-btn" onclick="CopyToClipboard()" title="Copies the content of the text area that has the focus">
    Copy to clipboard
  </button>
</div>
```

And the JavaScript code they execute:

```
let LockClosed = true;
let SwitchLockTargetByFocus = function () {
  LockClosed = !LockClosed;
  (function SwitchLockIcon() {
    //...
  })();
}

var IDWriterTargetName2 = 'inbox-textarea';

let SwitchCkeditorTextarea = function() {
  try {
    let textarea = document.getElementById(IDWriterTargetName2);
    ChangeFocusToTextArea(textarea.id);
  } catch(e) {
    ShowMessage(INVOX.MessageType.ERROR, e);
  }
}
```

```

}

let ChangeFocusToTextArea = function(id) {
  try {
    INVOX.SetTextWriter(INVOX.TextAreaTextWriter);
    INVOX.SetWriterTarget(id);
    //...
  } catch(e) {
    ShowMessage(INVOX.MessageType.ERROR, e);
  }
}

```

```

let CopyToClipboard = function() {
  try {
    let text = INVOX.GetText()
    //...
    navigator.clipboard.writeText(text)
    //...
  } catch(e) {
    ShowMessage(INVOX.MessageType.ERROR, e);
  }
}

```

How to test

You can test two possible scenarios: Local or Remote.

Login in the Local Dictation Service

1. Select “*Local*”.
2. Configure connection settings:
 - *Port*: indicates the port where the service is running.
3. Set user credentials.
4. Click on the Login button.



Dictation Service	Credentials
Connect to	Username
Local	Enter Username
Host	Password
localhost	Enter Password
Port	
8443	
	Log In

Figure 11: INVOX Medical Login Form

Login in the Remote Dictation Service

1. Select “*Remote*”.
2. Configure connection settings:
 - **Host:** indicates the host where the service is running.
 - **Port:** indicates the port where the service is listening.
3. Set user credentials.
4. Click on the Login button.

Test Simple Components

Introduction

INVOX Medical SDK allows you to create your own web components to provide feedback to the user.

Feedback about:

- Login Progress Status.
- Recognition Status.
- Session Status.
- Audio Permissions.
- Audio Input.
- Device Connection.
- User Information.
- Etc.

All of these simple components are implemented using the INVOX Medical API functions.

Components

1. **Progress Bar.** Displays the relative messages during the authentication process.
2. **Status Bar.** Displays messages about the session status.
3. **User Profile.** Displays a modal window with the data of the currently active user.
4. **Speech Recognizer Controls.**
 1. **Dictation On Button.** Allows to start speech recognition.
 2. **Dictation Off Button.** Allows to stop speech recognition.
 3. **Dictation Switch Button.** Allows to change the voice recognition status. Simplifies the use of the above buttons into one.

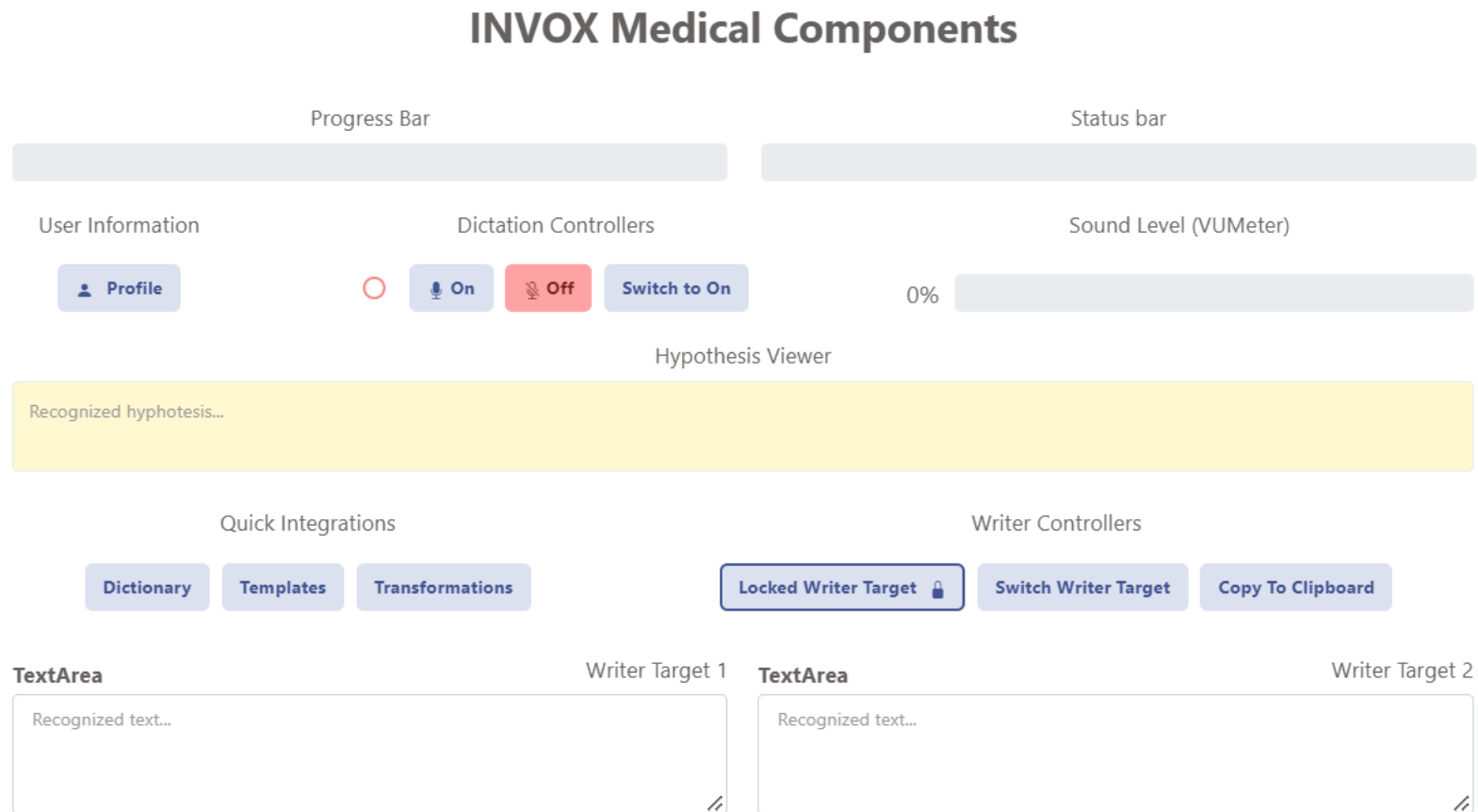


Figure 12: INVOX Medical SDK Sample about components creation

INVOX Medical Components

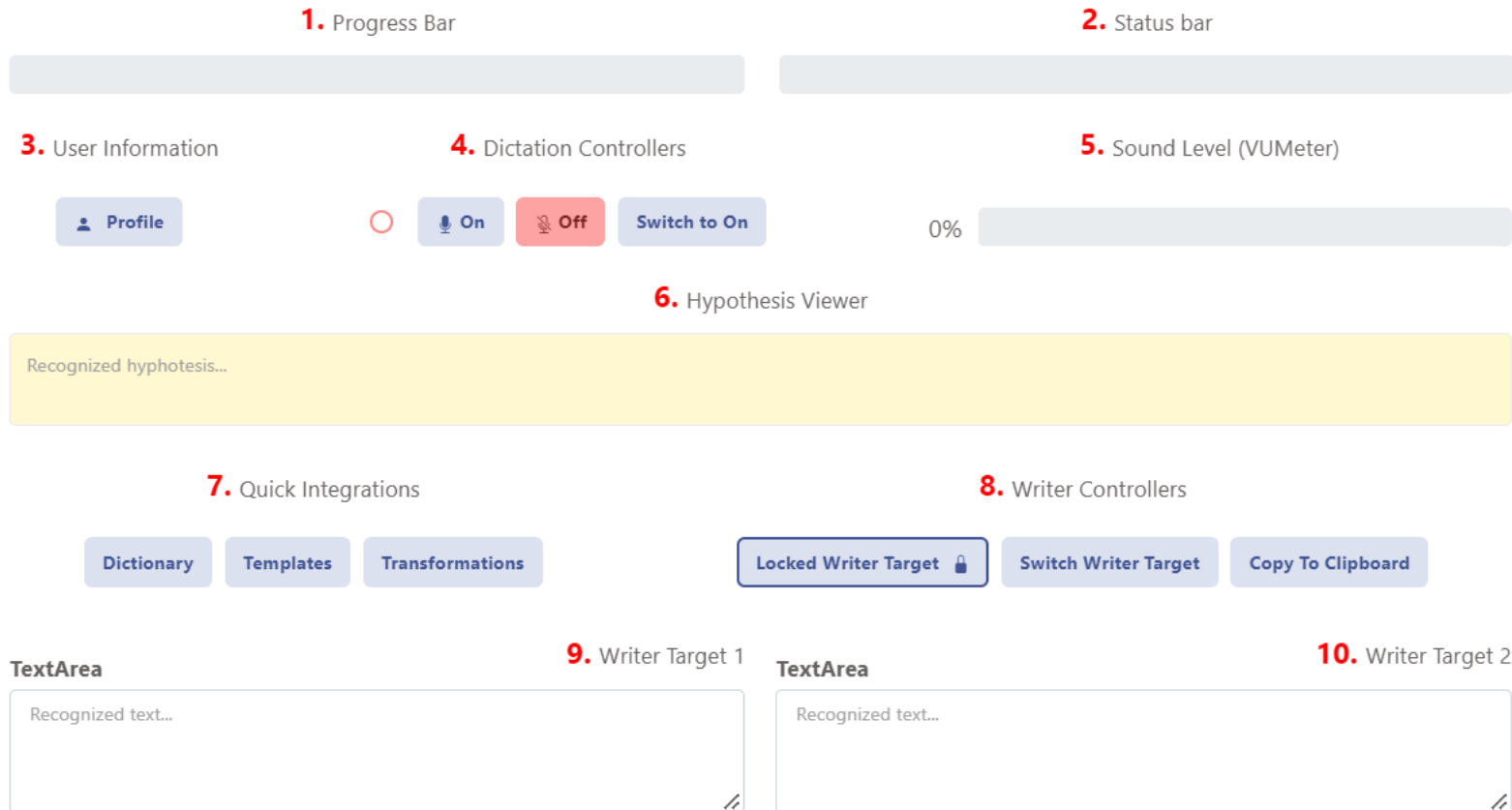


Figure 13: INVOX Medical SDK Components

5. **Input Audio Level.** Displays the audio level activity at all times.
6. **Hypothesis Viewer.** Displays the hypotheses of the recognised text during dictation, before displaying it in the corresponding WriterTarget.
7. **Quick Integrations.**
 1. **Dictionary.** Shows Dictionary view component.
 2. **Templates.** Shows Templates view component.
 3. **Transformations.** Shows Transformations view component.
8. **Writer Controllers.** Examples of focus control and other functions on the current WriterTarget.
 1. **Lock/unlock writer target.** To lock the Writer Target in the currently active text box.
 2. **Switch writer target.** Shall be used to change the focus of the WriterTarget, even if the previous lock button is active.
 3. **Copy to clipboard.** Shall be used to copy the contents of the current WriterTarget.
9. **Writer Target 1.** Is the first writing destination.
10. **Writer Target 2.** Is the second writing destination. It can be optional.

Code

You can find the complete code of this example referenced in the head of the `index.html` file:

- **main.js:** contains the components implementation.
- **style.css:** contains the styles of each component.

```
<!-- Page resources -->
<script type="text/javascript" src="main.js" charset="UTF-8"></script>
<link rel="stylesheet" href="style.css">
```

Progress bar

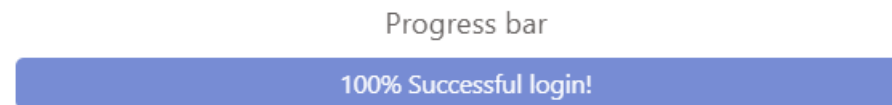


Figure 14: Complete progress bar component

HTML code to create the component:

```
<div class="progress invox-progressbar-container">
```

```

    <div id="inbox-progressbar" class="progress-bar inbox-progressbar-bar shadow-none" role="progressbar" aria-valuemin="0" aria-valuemax="100">
    </div>
</div>

```

To customize the behaviour of this progress bar and display the text shown in the image, the functions `INVOX.OnChangeProgressBar` and `INVOX.OnFinishProgressBar` have been redefined.

```

INVOX.OnChangeProgressBar(function (msg) {
    let HTMLProgressBar = document.getElementById("inbox-progressbar");
    let description = null;

    if (msg.Description !== undefined) {
        description = ` ${msg.Description}`;
    }
    HTMLProgressBar.innerHTML = `${+msg.Percent}% ${description}`;
    HTMLProgressBar.style.width = `${+msg.Percent}%`;
});

INVOX.OnFinishProgressBar(function () {
    let HTMLProgressBar = document.getElementById("inbox-progressbar");
    HTMLProgressBar.innerHTML = "100% Successful login!"
});

```

Status bar

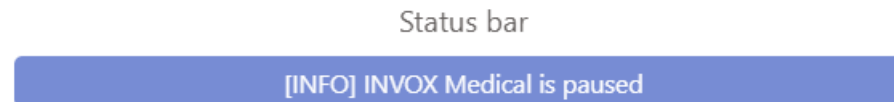


Figure 15: Status bar component

HTML code to create the component:

```

<input type="text" id="inbox-statusbar" class="form-control inbox-statusbar shadow-none" value="" readonly>

```

To customize the behaviour and appearance of this status bar and display the text shown in the image, the function `INVOX.OnChangeStatusBar` has been redefined.

```

INVOX.OnChangeStatusBar(function (msg, type) {

```

```

let HTMLStatusBar = document.getElementById("inbox-statusbar");
let statusBarInfoColor = getComputedStyle(document.documentElement).getPropertyValue('--inbox-highlight-normal-color');
let statusBarErrorColor = getComputedStyle(document.documentElement).getPropertyValue('--inbox-highlight-error-color');

HTMLStatusBar.value = `[${type}] ${msg}`;
HTMLStatusBar.style.backgroundColor = type == INVOX.MessageType.ERROR ? statusBarErrorColor : statusBarInfoColor;
});

```

User information

The most important part of this section is the use of the function `INVOX.GetUserInfo` that allows to obtain all the information of the active user.

Once this information is obtained, it can be shown to the user, for example, through a modal form.

The HTML code for the button and the modal form is shown below.

```
<!-- Button -->
```

```

<button id="inbox-user-info-btn" class="btn inbox-base-btn"
  onclick="ShowUserInfo()">
  Show
  <i class="bi bi-person-fill inbox-user-info-icon"></i>
</button>

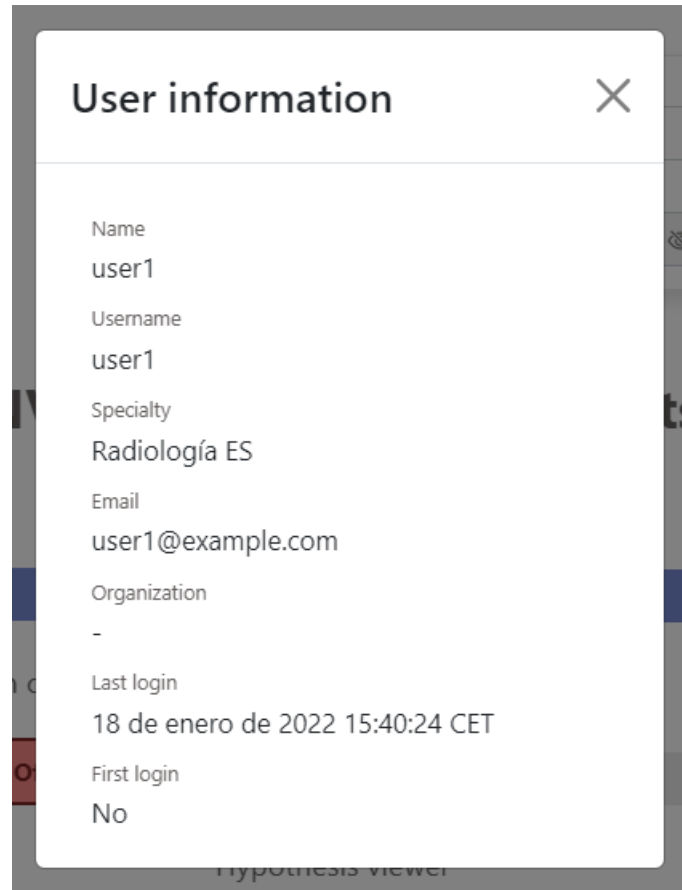
```

```
<!-- Modal form where user information is displayed -->
```

```

<div id="inbox-modal-user-info" class="modal fade" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel">
  <div class="modal-dialog modal-sm">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="staticBackdropLabel">User information</h5>
        <button class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <label class="inbox-modal-user-info-label">Name</label>
        <input type="text" id="inbox-modal-user-info-name" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Name">
        <label class="inbox-modal-user-info-label">Username</label>
        <input type="text" id="inbox-modal-user-info-login" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Username">
        <label class="inbox-modal-user-info-label">Specialty</label>
      </div>
    </div>
  </div>
</div>

```

A modal window titled "User information" with a close button (X) in the top right corner. The form displays user details for "user1".

Name	user1
Username	user1
Specialty	Radiología ES
Email	user1@example.com
Organization	-
Last login	18 de enero de 2022 15:40:24 CET
First login	No

Figure 16: User information form


```

        <input type="text" id="inbox-modal-user-info-specialty" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Specialty">
        <label class="inbox-modal-user-info-label">Email</label>
        <input type="text" id="inbox-modal-user-info-email" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Email">
        <label class="inbox-modal-user-info-label">Organization</label>
        <input type="text" id="inbox-modal-user-info-org" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Organization">
        <label class="inbox-modal-user-info-label">Last login</label>
        <input type="text" id="inbox-modal-user-info-lastlogin" class="form-control inbox-modal-user-info-text" readonly="" placeholder="Last login">
        <label class="inbox-modal-user-info-label">First login</label>
        <input type="text" id="inbox-modal-user-info-isfirstlogin" class="form-control inbox-modal-user-info-text" readonly="" placeholder="First login">
    </div>
</div>
</div>
</div>

```

And the JavaScript code, which obtains the user's data and displays it on the screen, is as follows:

```

let ShowUserInfo = function () {
    try {
        let HTMLModalUserInfo = document.getElementById("inbox-modal-user-info");
        if (!HTMLModalUserInfo) {
            throw "Element \'inbox-modal-user-info\' not found";
        }
        let userInfo = INVOX.GetUserInfo()
        document.getElementById("inbox-modal-user-info-name").value = userInfo.Login || '-';
        document.getElementById("inbox-modal-user-info-login").value = userInfo.Name || '-';
        document.getElementById("inbox-modal-user-info-specialty").value = userInfo.Specialty || '-';
        document.getElementById("inbox-modal-user-info-email").value = userInfo.Email || '-';
        document.getElementById("inbox-modal-user-info-org").value = userInfo.Organization || '-';
        document.getElementById("inbox-modal-user-info-lastlogin").value = userInfo.LastLogin || '-';
        document.getElementById("inbox-modal-user-info-isfirstlogin").value = userInfo.IsFirstLogin ? 'Yes' : 'No' || '-';

        let modal = new bootstrap.Modal(HTMLModalUserInfo, {});
        modal.show();
    } catch (e) {
        console.error('ERROR: ' + e);
    }
}

```

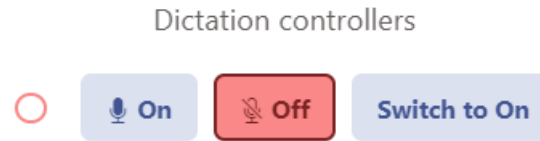


Figure 17: Dictation controllers

Speech recognizer controls

This group of buttons belongs to the group of “Dictation controllers”, for example the buttons that manage the actions of the recognizer.

- **On.** This button starts the recognizer, leaving INVOX Medical listening. Execute `INVOX.SetDictationRunning`.
- **Off.** This button stops the recognizer, leaving INVOX Medical paused. Execute `INVOX.SetDictationPaused`.
- **Switch to On.** This button changes the status of the recognizer, so that if it is paused it switches to listening, or vice versa. Execute `INVOX.SwitchDictation`.

The HTML code for the implementation of these buttons is shown below:

```
<div>
  <button id="invox-btn-dictation-on" class="btn invox-dictation-btns invox-on-btn" onclick='OnDictationRunning()'>
    <i class="bi bi-mic"></i> On
  </button>
  <button id="invox-btn-dictation-off" class="btn invox-dictation-btns invox-off-btn invox-off-focus-btn" onclick='OnDictationPaused()'>
    <i class="bi bi-mic-mute"></i> Off
  </button>
  <button id="invox-btn-switch-dictation" class="btn invox-dictation-btns invox-on-btn" onclick='OnDictationSwitch()'>Switch to On</button>
</div>
```

And the JavaScript code that implements its functionality:

```
function OnDictationPaused () {
  try {
    INVOX.SetDictationPaused();
  } catch(e) {
    console.error('ERROR: ' + e);
  }
}

function OnDictationRunning () {
  try {
```

```

        INVOX.SetDictationRunning();
    } catch(e) {
        console.error('ERROR: ' + e);
    }
}
function OnDictationSwitch () {
    try {
        INVOX.SwitchDictation();
    } catch(e) {
        console.error('ERROR: ' + e);
    }
}

```

Audio level indicator

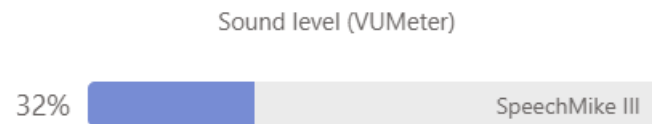


Figure 18: Audio level indicator component

HTML code to create the component:

```

<div class="d-flex align-items-center">
  <span id="inbox-vumeter-value" class="inbox-vumeter-value">32%</span>
  <div class="inbox-progress-vumeter-container">
    <div id="inbox-progress-vumeter" class="progress-bar inbox-progress-vumeter-bar" role="progressbar" aria-valuemin="0" aria-valuemax="100">
      <span id="inbox-microphone-name" class="inbox-microphone-name"></span>
    </div>
  </div>
</div>

```

To customize the behaviour, the function `INVOX.OnUpdateVumeterMicActivity` has been redefined, which updates the width of the audio indicator according to the percentage of microphone activity detected.

```

INVOX.OnUpdateVumeterMicActivity(function (activity) {

    let HTMLVUMeter = document.getElementById("inbox-progress-vumeter");
    let HTMLVUMeterValue = document.getElementById("inbox-vumeter-value");

```

```
HTMLVUMeter.style.width = `${parseInt(activity)}%`;
HTMLVUMeterValue.innerText = `${parseInt(activity)}%`;
});
```

In addition, information about the detected microphone has been included in the component. For this purpose, the `INVOX.GetMicrophoneName` function is used after securing the microphone permissions, the functions of which are discussed below.

Hypothesis Viewer

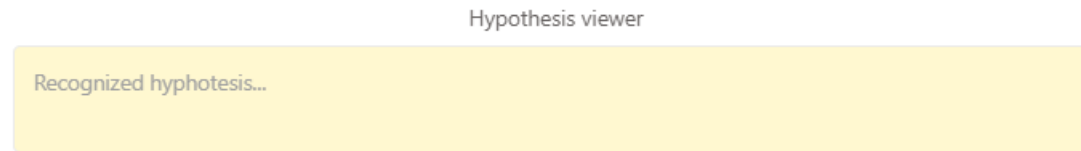


Figure 19: Hypothesis Viewer component

HTML code to create the component:

```
<textarea id="invoy-hyphotesis-viewer" class="invoy-hypothesis-viewer shadow-none" placeholder="Recognized hyphotesis..." readonly rows="3"></te
```

To customize the behaviour, the function `INVOX.OnChangeVisorHypothesis` has been redefined, which writes all hypotheses detected by the recognizer to the same destination.

```
INVOX.OnChangeVisorHypothesis(function (msg, dictationEventType) {

    let dictationType = {};
    let HTMLVisor = document.getElementById("invoy-hyphotesis-viewer");
    dictationType[INVOX.dictationEventType.ACCEPTED] = 'ACCEPTED';
    dictationType[INVOX.dictationEventType.REJECTED] = 'REJECTED';
    dictationType[INVOX.dictationEventType.PARTIAL] = 'PARTIAL';
    dictationType[INVOX.dictationEventType.COMMAND] = 'COMMAND';
    dictationType[INVOX.dictationEventType.MACRO] = 'MACRO';

    HTMLVisor.innerText = `[${dictationType[dictationEventType]}] ${msg}`;

});
```

Writer controllers

The Writer controllers implement very simple functions that allow you to easily manage the current `WriterTarget`.

Writer controllers

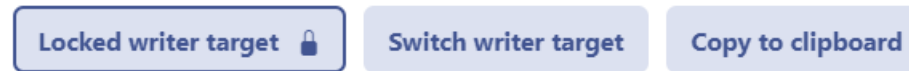


Figure 20: Writer controllers

- **Locked writer target.** With the button active the text recognised by INVOX Medical will be written in the current Writer Target (text box with the focus), even if you manually switch the focus to another editor.
- **Switch writer target.** Switches the focus between the WriterTargets set.
- **Copy to clipboard.** Gets the text from the current Writer Target (textbox with the focus) and copies it to the clipboard.

The HTML code to create the buttons is as follows:

```
<div class="d-flex flex-wrap justify-content-center">
  <button id="inbox-switch-lock-btn" class="btn inbox-base-btn inbox-switch-lock-btn inbox-switch-lock-focus-btn" onclick="SwitchLockTargetByFocus()">
    <span id="inbox-switch-lock-btn-text">Locked writer target</span>
    <i id="inbox-lock-icon" style="margin-left: 5px; font-size: 1em;" class="bi bi-lock-fill"></i>
  </button>
  <button id="inbox-switch-writer-btn" class="btn inbox-base-btn" onclick="SwitchTextAreas()" title="Switch focus between text areas.">
    Switch writer target
  </button>
  <button id="inbox-clipboard-btn" class="btn inbox-base-btn" onclick="CopyToClipboard()" title="Copies the content of the text area that has focus">
    Copy to clipboard
  </button>
</div>
```

And the JavaScript code they execute:

```
let LockClosed = true;
let SwitchLockTargetByFocus = function () {
  LockClosed = !LockClosed;
  (function SwitchLockIcon() {
    //...
  })();
}
```

```

var IDWriterTargetName1 = 'inbox-textarea-1';
var IDWriterTargetName2 = 'inbox-textarea-2';

let SwitchTextAreas = function () {
    let target1 = document.getElementById(IDWriterTargetName1);
    let target2 = document.getElementById(IDWriterTargetName2);
    if (target1.classList.contains(ClassWriterTargetFocused)) {
        ChangeFocusToTextArea(target2.id);
    } else {
        ChangeFocusToTextArea(target1.id);
    }
}

let ChangeFocusToTextArea = function(id) {
    try {
        INVOX.SetTextWriter(INVOX.TextAreaTextWriter);
        INVOX.SetWriterTarget(id);
        //...
    } catch(e) {
        ShowMessage(INVOX.MessageType.ERROR, e);
    }
}

let CopyToClipboard = function() {
    try {
        let text = INVOX.GetText()
        //...
        navigator.clipboard.writeText(text)
        //...
    } catch(e) {
        ShowMessage(INVOX.MessageType.ERROR, e);
    }
}

```

Microphone permissions

In this example page, two behaviours have been defined when microphone permissions are accepted or rejected.

The function `INVOX.OnGrantedAudioSource` will display in the audio prompt the name of the microphone detected at login. The JavaScript code that implements this is as follows:

```

INVOX.OnGrantedAudioSource(function () {
    let HTMLMicrophoneName = document.getElementById("invox-microphone-name");
    let micName = INVOX.GetMicrophoneName();
    HTMLMicrophoneName.innerHTML = micName;
    HTMLMicrophoneName.title = `Using microphone: ${micName}`;
});

```

If on the other hand the microphone permissions are rejected then the recognition will be stopped if active and the microphone name will be cleared. The function `INVOX.OnDeniedAudioSource` defines this behaviour:

```

INVOX.OnDeniedAudioSource(function () {
    INVOX.SetDictationPaused();
    let HTMLMicrophoneName = document.getElementById("invox-microphone-name");
    HTMLMicrophoneName.innerHTML = "You need to granted audio permissions";
});

```

How to test

You can test two possible scenarios: Local or Remote.

Login in the Local Dictation Service

1. Select “**Local**”.
2. Configure connection settings:
 - **Port:** indicates the port where the service is running.
3. Set user credentials.
4. Click on the Login button.

Login in the Remote Dictation Service

1. Select “**Remote**”.
2. Configure connection settings:
 - **Host:** indicates the host where the service is running.
 - **Port:** indicates the port where the service is listening.
3. Set user credentials.
4. Click on the Login button.



Dictation Service	Credentials
Connect to	Username
<div>Local</div>	<div>Enter Username</div>
Host	Password
<div>localhost</div>	<div>Enter Password</div>
Port	
<div>8443</div>	<div>Log In</div>

Figure 21: INVOX Medical Login Form