Ausgabe 5.17 August/September www.phpmagazin.de S&S

PHPmagazin

PHPmagazin

Deutschland 9,80 € Österreich 10,80 € | Schweiz 19,20 sFr
Niederlande 11,25 € | Luxemburg 11,25 €

PHP • JavaScript • Open Web Technologies

Sail away!

Echtzeitfähige Applikationen mit Sails und Angular

Router

Alternative PHP-Router im Überblick

Jenkins und Angular

Automatisch, praktisch, gut

GitHub-Add-ons

Eine Schippe drauflegen

Zepto.js

jQuery auf Diät

4 195649 609809 05

Verknüpfung eines Anrufs mit der Limbas-Datenbank

Anrufmonitor in Limbas

Limbas ist ein Open-Source-PHP-Framework, das die Umsetzung von individuellen Datenbankanwendungen ermöglicht. Durch den modularen Aufbau, die SOAP-Schnittstelle und die Möglichkeit, es mit eigenen Skripten zu erweitern, lassen sich neue Features leicht in Limbas integrieren. Ein praktisches Beispiel ist die Anbindung der AVM FRITZ!Box an Limbas, um den Namen oder die Nummer eines Anrufers in Echtzeit im Browser anzuzeigen, Anrufe zu loggen oder automatisch mit einem bestehenden Datensatz zu verknüpfen. In diesem Artikel soll gezeigt werden, wie diese Funktionalität implementiert und an Limbas angebunden werden kann.

von Peter Greth

Um die gesamte Funktionalität umzusetzen, werden insgesamt fünf Dateien an unterschiedlichen Stellen hinzugefügt: Die Dateien *callmonitor.php*, *callserver.php* und *CallList.class.php* befinden sich auf einem Server, der sich sowohl mit der FRITZ!Box als auch mit dem Limbas-Server verbinden kann. Man kann die Dateien aber auch direkt auf dem Limbas-Server platzieren. Auf diesem werden des Weiteren die Dateien *ext_multiframe.inc* und *ext_main.js* im Limbas-Extension-Verzeichnis *(<LimbasVerzeichnis>/dependent/EXTENSIONS/)* oder einem beliebigen Unterordner erstellt. Der gesamte Beispielcode kann im Limbas-Wiki unter [1] eingesehen werden, Limbas selbst kann unter [2] kostenlos heruntergeladen oder auch online getestet werden.

Der FRITZ!Box Callmonitor

Der Callmonitor einer FRITZ!Box ist ein Dienst, der Informationen über ein- und ausgehende Anrufe zur Verfügung stellt. Damit er genutzt werden kann, muss einmalig auf einem verbundenen Telefon #96*5* gewählt werden, bzw. #96*4*, um ihn wieder zu deaktivieren. Wurde der Dienst aktiviert, werden auf TCP Port 1012 Informationen zu aktuellen Anrufen bereitgestellt, wobei jeder Anruf in mehrere Ereignisse aufgeteilt wird: Klingelt beispielsweise bei einem eingehenden Anruf das Telefon, wird das Ereignis *RING* ausgelöst, ein ausgehender Anruf wird *CALL* genannt. *CONNECT* wird nur ausgelöst, wenn der angerufene Teilnehmer abhebt und so die Verbindung zustande kommt. Wird das Telefonat im Anschluss beendet, wird ein *DISCONNECT* gesendet.

```
[root@phone callmonitor]# php callmonitor.php
10.04.17 10:38:32;RING;0;0891234567;498949026850;SIP3;
10.04.17 10:38:35;CONNECT;0;4;0891234567;
10.04.17 10:38:59;DISCONNECT;0;24;
10.04.17 10:41:42;CALL;0;SIP3;498949026850;0177123456;
10.04.17 10:41:45;CONNECT;0;4;0177123456;
10.04.17 10:43:51;DISCONNECT;0;126;
10.04.17 10:44:10;RING;0;0177123456;498949026850;SIP3;
```

Abb. 1: Output der FRITZ!Box, passend zu Ereignissen in Abbildung 2

Für jedes dieser Ereignisse sendet die FRITZ!Box eine Zeichenkette (**Abb. 1**) mit zusätzlichen Informationen wie Datum und Uhrzeit, anrufende/angerufene Nummer bzw. Nebenstelle, Dauer des Gesprächs in Sekunden (nur bei *DISCONNECT*) oder Connection-ID. Letztere wird verwendet, um einen Anruf über mehrere Ereignisse hinweg identifizieren zu können.

Auswertung des Callmonitors

Um diese Funktionalität der FRITZ!Box zu nutzen, wird ein Dienst benötigt, der dauerhaft ausgeführt wird, die bereitgestellten Daten auswertet, sich vom Limbas-Server zusätzliche Informationen holt und an alle verbundenen Limbas-Clients weitergibt. Diese Aufgabe ist in der Datei *CallMonitor.php* (Listing 1) implementiert. Über den jetzt offenen Port 1012 wird zunächst in PHP eine Socket-Verbindung mit der FRITZ!Box hergestellt. Die erhaltenen Anrufinformationen werden dann mithilfe der Klasse *CallList* verwaltet. Die *CallList* übernimmt die Aufgabe, mehrere Ereignisse einem Anruf durch die Connection-ID zuzuordnen und die Daten der letzten zehn Anrufe zu speichern. Die Anzahl der gespeicherten Anrufe lässt sich jedoch leicht im Code ändern.

Für jedes erhaltene Ereignis wird über die SOAP-Schnittstelle eine Anfrage an den Limbas-Server gesendet, die die aktuell gesammelten Daten des Telefonats, z. B. Telefonnummer oder Gesprächsdauer, enthält. Als

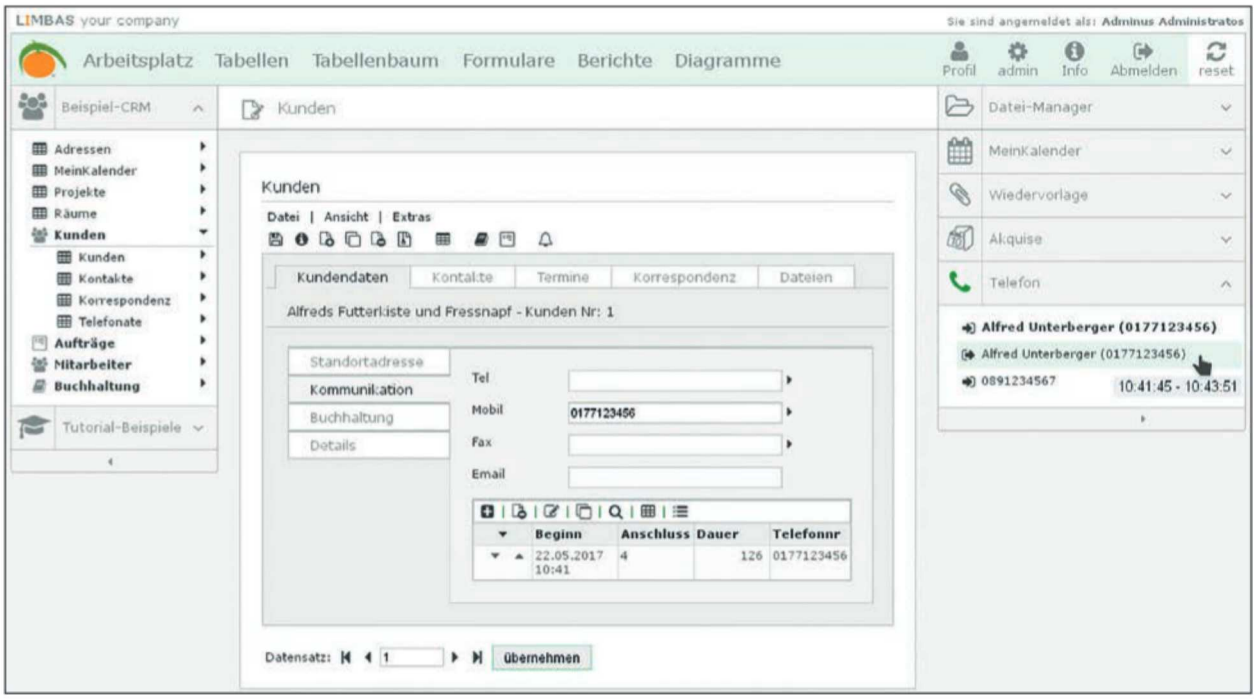

Abb. 2: Limbas im Browser

Rückgabewert der Anfrage wird ein HTML-String erwartet. Was dieser beinhaltet, kann der Serveradministrator durch die Limbas-Extensions selbst entscheiden und implementieren. Eine mögliche Implementierung wird in Abschnitt „Beispielimplementierung der SOAP-Schnittstelle“ beschrieben.

Zu jedem Anruf gibt es dadurch eine HTML-Beschreibung, die auch in der Call List gespeichert wird. Die Beschreibungen der letzten Anrufe müssen nun nur noch an alle verbundenen Limbas-Clients gesendet werden. Da der Callmonitor dauerhaft auf eintreffende Ereignisse der FRITZ!Box wartet und somit nicht in der Lage ist, mehrere Clients zu verwalten, wird ein weiterer Daemon benötigt.

Verteilung der Informationen an mehrere Clients

Der Call Server (*callserver.php*) verwaltet in PHP mehrere WebSocket-Verbindungen von Limbas-Clients sowie die Verbindung mit dem Callmonitor. Erhält er von diesem eine Nachricht mit den HTML-Beschreibungen der letzten Anrufe, leitet er diese Nachricht an alle verbundenen Clients weiter. Damit neue Verbindungen sofort Anrufinformationen erhalten und nicht auf das Senden eines neuen Ereignisses warten müssen, wird die

Listing 1

Datei: callmonitor.php

```
...

# connect to fritzbox
$fritzboxSocket = fsockopen($fritzHost, $fritzPort);

# store last 10 calls
$callList = new CallList(10);

# wait for new lines
while(true) {
  $newLine = fgets($fritzboxSocket);
  if($newLine != null) {
    echo $newLine;
    handleFritzboxEvent($newLine);
  } else {
    sleep(1);
  }
}
```

```
function handleFritzboxEvent($line) {
  ...
  # split into parts
  $parts = split(';', $line);

  # extract data
  $dateTime = $parts[0];
  $type = $parts[1];
  $connectionId = $parts[2];

  # differ between event types
  if($type == 'RING') { # incoming ring
    $callerNr = $parts[3];

    # store phone nr
    $callList->onRing($connectionId, $callerNr);
  }
  ...
  limbasSocketNotification($connectionId, $type);
}

function limbasSocketNotification($cid, $type) {
  ...
  # Get data from limbas via soap given call data
  $data = null;
  $lmpar[0]['extension']["OnPhone{$type}"] = json_
                    encode($callList->get($cid));
  $cdata = call_client($lmpar);
  if(count($cdata) > 0 && $cdata[0]){
    $data = $cdata[0];
  }

  ...
  # Notify socket server
  sendDataToWebsocketServer($wsHost, $wsPort,
                    json_encode($data));
  ...
}
```

letzte Nachricht des Callmonitors zusätzlich im Call Server zwischengespeichert.

Integration in das Limbas Frontend

Um das Feature zu vervollständigen, wird letztendlich noch eine Instanz benötigt, die die Verbindung zum Call Server herstellt und dessen Informationen in Limbas darstellt. Limbas kann durch verschiedene Skripte erweitert werden. Welche das sind, ist im Wiki unter [3] dokumentiert.

In Erweiterungsdateien mit dem Namen *ext_multiframe.inc* (Listing 2) können neue Elemente zum Multiframe, dem Menü an der rechten Seite, hinzugefügt werden. Das erstellte Element wird mit einem Telefon-Icon dargestellt und kann auf- bzw. zugeklappt werden (**Abb. 2**), wobei jedes Mal eine selbst definierte JavaScript-Funktion ausgeführt wird. Diese wird in *ext_main.js* implementiert, stellt zunächst die Verbindung mit dem Call Server her und zeigt bei eintreffenden Nachrichten die mitgesendeten Beschreibungen der Anrufe im erstellten Multiframe-Element an.

Um dem Nutzer Einsicht in den Status der Verbindung zum Call Server zu geben, ändert sich die Farbe des Telefon-Icons: Ist das Multiframe-Element zugeklappt, wird keine Verbindung aufgebaut (grau). Ist das Element aufgeklappt, steht rot für eine fehlgeschlagene Verbindung, gelb für einen Verbindungsversuch und grün für eine bestehende Verbindung. Bei einer neuen Nachricht, also einer Aktualisierung der Anrufe, wird das Icon kurzzeitig blau.

Beispielimplementierung der SOAP-Schnittstelle

Wie bereits beschrieben, sendet der Callmonitor eine SOAP-Anfrage an den Limbas-Server. Die Schnittstelle kann dabei vom Administrator in der Extension-Datei *ext_soap.inc* (Listing 3) implementiert werden. Es werden vier Funktionen erwartet: *extSoapOnPhoneRing*, *extSoapOnPhoneCall*, *extSoapOnPhoneConnect* und *extSoapOnPhoneDisconnect*. Je nach ausgelöstem Ereignis wird dabei die passende Funktion aufgerufen.

In der Demodatenbank kann diese Funktionalität beispielsweise genutzt werden, um direkt den Namen und den Link zum Datensatz des anrufenden Kunden/Kontakts anzuzeigen sowie Zeitpunkt und Dauer des Gesprächs in einer eigenen Tabelle zu loggen und direkt mit dem Kunden zu verknüpfen. In der Limbas-Demodatenbank gibt es eine Tabelle „Kunden“ mit Feldern wie ID, Firmenname oder Telefonnummer. Diese ist über eine 1:n-Verknüpfung mit der Tabelle „Kontakte“ verknüpft, in der beispielsweise der Name des Kontakts oder dessen Telefonnummer gespeichert sind.

Um den passenden Kunden zur anrufenden Nummer zu finden, wird

Listing 2

Datei ext_multiframe.inc

```
$melements["id"] = "Phone";
$melements["name"] = "Telefon";
$melements["link"] = "";
$melements["target"] = "main";
$melements["preview"] = "";
$melements["event"] = "limbas_
                    callmonitor_toggle";
$melements["params"] = "";
$melements["gicon"] = "lmb-phone";
$melements["autorefresh"] = 1;
$menu[0][] = $melements;
```

Listing 3

Datei: ext_soap.inc

```
# display name and 'outgoing'-icon
function extSoapOnPhoneCall($param_string=null, $param_
                            array=null, $lmb=null) {
 return getCredentials($param_string, 'lmb-sign-out');
}

# display name and 'ingoing'-icon (bold)
function extSoapOnPhoneRing($param_string=null, $param_
                            array=null, $lmb=null) {
 return '<b>' . getCredentials($param_string, 'lmb-sign-in') .
                                                '</b>';
}

...

function getCredentials($param_string, $imgClass) {
 # decode input
 $decoded = json_decode($param_string, 1);

 # get phone nr
 $phoneNr = $decoded['phoneNr'];

 # ensure data is stored in session
 if(!$_SESSION['callmonitor']) { $_SESSION['callmonitor'] =
                                          array(); }
 if(!$_SESSION['callmonitor']['nr'.strval($phoneNr)]) {
  # store customer data in session
  $_SESSION['callmonitor']['nr'.strval($phoneNr)] =
                getCustomerDataByPhoneNr($phoneNr);
 }

 # get date
 if($decoded['startDate'] != null) {
  # extract start time
  $dateObj = DateTime::createFromFormat('d.m.y H:i:s',
                                   $decoded['startDate']);
  $toolTip = $dateObj->format('H:i:s');

  # add end time on disconnect
  if($decoded['lengthSec'] != null) {
   $dateObj->modify("+{$decoded['lengthSec']} seconds");
   $toolTip .= ' - ' . $dateObj->format('H:i:s');
  }
 } else {
  $toolTip = 'Anruf';
 }

 # return customer data to callmonitor
 $customerData = $_SESSION['callmonitor']['nr'.
                                   strval($phoneNr)];
 if($customerData) {
  ob_start();
  pop_menu2("{$customerData['contactName']} ($phoneNr)",
            "$toolTip", null, $imgClass, null, "parent.main.
             location.href='main.php?action=gtab_change&
            gtabid=4&ID={$customerData['customerId']}';");
  return ob_get_clean();
 } else {
  ob_start();
  pop_menu2("$phoneNr", "$toolTip", null, $imgClass, null,
                                                 null);
  return ob_get_clean();
 }
}
```

zunächst die Tabelle „Kontakte“ durchsucht. Wurde eine Kontaktperson gefunden, kann über die Verknüpfung direkt der zugehörige Kunde herausgefunden werden. War die Suche erfolglos, wird zusätzlich noch die Kundentabelle durchsucht. Wurde ein Kunde gefunden, kann ein Link generiert werden, der den Nutzer direkt zum Detaildatensatz des Kunden führt. Zusätzlich wird ein Icon angezeigt, das kennzeichnet, ob der Anruf ein- oder ausgehend ist. In der Funktion *extSoapOnPhoneDisconnect* ist zusätzlich das Speichern des Anrufs sowie das Verknüpfen mit dem Kundendatensatz implementiert.

In **Abbildung 2** sieht man beispielsweise auf der rechten Seite im Telefonelement, dass zuerst eine unbekannte Nummer angerufen hat (unterste Zeile). Beim zweiten Telefonat (mittlere Zeile) wurde zur angerufenen Nummer der Kontakt „Alfred Unterberger“ gefunden. Durch das veränderte Icon lässt sich erkennen, dass es sich diesmal um einen ausgehenden Anruf handelt. Als neuestes Ereignis (oberste Zeile) wird ein eingehender Anruf desselben Kontakts angezeigt. Da in diesem Szenario noch nicht abgehoben wurde, sondern das Telefon noch klingelt, wird der Text fett dargestellt.

Klickt man auf einen Eintrag des Elements, wird im Limbas-Hauptfenster der zugehörige Detaildatensatz des Kunden (hier „Alfreds Futterkiste“) angezeigt. In diesem ist auch der vorherige Anruf des Kontakts zu sehen.

Fazit

Durch die Erweiterbarkeit des Limbas-Frameworks konnte der Callmonitor an Limbas angebunden werden. Die Funktionalität wurde mithilfe der FRITZ!Box gezeigt, es ist aber auch möglich, jede andere Telefonanlage in das System zu integrieren, sofern sie eine geeignete Schnittstelle bereitstellt. Es wurden die Funktionsweise der Anbindung sowie eine mögliche Implementierung der Schnittstelle beschrieben. Welche Informationen letztendlich bei einem Anruf dargestellt werden sollen, kann jeder Limbas-Entwickler selbst entscheiden. Somit ist die Callmonitor-Funktionalität – wie auch Limbas – nicht nur auf ein Szenario begrenzt, sondern auf viele Wünsche anpassbar.

Peter Greth studiert Informatik an der TU München und arbeitet nebenbei bei der Limbas GmbH.

pgreth@limbas.com

Links & Literatur

[1] www.limbas.org/wiki/Callmonitor

[2] www.limbas.com

[3] www.limbas.org/wiki/Skript-Erweiterungen