Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

ARC: an Alexa based suite of skills for Residential Care services

Josu Pastor Almor

Director: J. Daniel Prades
Realitzat a: Departament d’Enginyeries: Electrònica

Barcelona, 18 de maig de 2017

CONTENTS

Chapter 1

INTRODUCTION

1.1 Amazon Echo and Alexa

Figure 1: Amazon Echo device

For decades, science fiction movies and novels have showed us what the future smart home could be like and, particularly, artificial intelligence (AI) home assistants.

In 2017 different companies have attempted to create their own take on these assistants and the result is a reduced, though varied, spectrum of solutions.

Google makes the Google Home, a "smartspeaker" that stands out for its Google Assistant implementation and search algorithms. Microsoft has Cortana, a virtual AI assistant included in every copy of Windows 10. Even Apple has Siri and plans on producing a device of their own.

In 2015 Amazon introduced its own take on the matter, the Amazon Echo (shown in figure 1). It's a Bluetooth speaker that has a built-in microphone and connects to the Amazon Web Services (AWS) through Wi-Fi.

Specifically, it connects to the voice-controlled AI assistant service Alexa, which also happens to be the device’s wake word (i.e. the word used to start an interaction with it). This allows a user to speak to Alexa directly, and ask her to play music, get weather information and news, set alarms, control lights around the house and tell terrible jokes, amongst other things.

But what truly sets the Echo apart from its competition is the ability to learn new Skills. Skills work like apps on smartphones, they are made by developers and can be downloaded from the Skill store through the Alexa phone app. Contrary to traditional apps however, Skills have no visual interface and the interaction relies completely on voice. This has very interesting implications when it comes to defining a user interface.

Alexa Residential Care (ARC) is a suite of skills designed to provide residential care services to seniors through voice.

It is the first phase of a larger project still in early development, and so it must be viewed not as production ready-software, but as a proof of concept for the set of ideas put forth by my TFG supervisor, Dr. J. Daniel Prades, as well as his German college from the Braunschweig University of Technology, Prof. Andreas Waag.

1.2 Motivation

When I first learned about this project I was very intrigued, having personally seen a family member go through their latter stages of life and slowly making it evident that even the most capable and independent of human beings is not immune to the nature of life decay. Memory loss, confusion, lack of technological know-how and loneliness, amongst others, are not trivial matters and we all have people in our lives that suffer from these conditions in one way or another.

With this in mind, I was moved by the possibility of creating a set of tools with the objective of helping out people in such need of assistance on their day to day activities.

On top of that, if we add the novelty of the interface design and the fact that a project of this nature requires a healthy mix of technologies and design philosophies then we end up with a very rich and interesting venture.

1.3 Objectives

Alexa Residential Care (ARC) is a suite of skills designed to provide residential care services to seniors through voice.

For voice interaction, the Amazon Echo platform and its assistant, Alexa, will be used.

In a first phase, 4 main skills will be implemented and tested in English.

- Skill 1: Medication reminder and dose explainer.
- Skill 2: Remote emergency notification.
- Skill 3: Text message dictating.
- Skill 4: AI conversational module.

Additionally, the skills should feature:

- Logging of all interactions, for supervision.
- Implemented in English, but the code should be ready for bulk translation into other languages. Currently Alexa is available in English (UK, USA) and German.
- Implementation over existing calendar platform (e.g. Google Services API).

Finally, an interface for ARC management might be necessary

1.4 Technological Context

1.4.1 Alexa Skills Kit

The Alexa Skills Kit (ASK) is a collection of APIs, tools, documentation, and code samples that allow a developer to build custom Skills for the Alexa platform.

The Skills can be made using any programming language, but in particular Alexa developers tend to use either node.js or Python.

For this project, Python 2.7 was chosen due to the availability of Flask-ASK and the developer having had experience using Flask.

1.4.2 Flask and Flask-ASK

Flask[1] is a microframework for Python based on Werkzeug, a WSGI utility, and Jinja 2, a templating engine for Python.

Flask-ASK[2] is an extension to Flask that does the following:

- Has decorators to map Alexa requests and intent slots to view functions.
- Helps construct ask and tell responses as well as reprompts.
- Makes session management easy.
- Allows for the separation of code and speech through Jinja templates
- Verifies Alexa request signatures.

Of particular interest was the ability to use decorators to handle requests made by the user. This is explained in detail in the following section.

1.4.3 Using Alexa

Voice interaction with Alexa can be distilled into the following components:

- Utterances: an utterance is a phrase uttered by the user.
- Intents: the intent is the interpreted intention of the user's utterance.
- Slots: a slot is specific type of information uttered by the user, such as a date, number, or anything that the developer decides to use. The distinction is made in order to easily strip the utterance into relevant bits of information for Alexa to process.

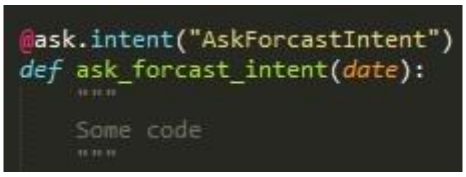
So for instance in the following interaction:

Human: Alexa, how's the weather today?

The components would be:

- **Utterance**: "Alexa, how's the weather today?"
- **Intent**: AskForecastIntent. This is an intention defined by the developer that deals with the problem of requesting the weather forecast information.
- **Slots**: today. This is datatype data that takes the value of today's date.

So using Flask-ASK we can map the utterance to a method in python thanks to the intent, as shown in figure 2:



```python
@ask.intent("AskForcastIntent")
def ask_forcast_intent(date):
    """
    Some code
    """
```

Figure 2:Python Flask-ASK intent method

This allows the development of python based skills with methods that are mapped to user intents.

The Echo device is not in charge of making such distinctions, however. It merely listens to the user and sends the utterance to the Alexa cloud service hosted by Amazon, where it is distilled into the components mentioned above. We will go into detail of how this is achieved further on.

1.4.4 Echosim.io

Echosim.io[3] is an Alexa skill testing tool provided by iQuarius Media. Using a microphone connected to a computer and logging into our Amazon account, we can test a custom skill using this site without having to purchase an Amazon Echo device. To use it, however, one has to click and hold a button instead of speaking to it with a wake word, which is cumbersome by comparison.

Lucky for us we got our hand on an Echo not long into development so we didn't need to use Echosim.io more than we had to.

1.5 Kanban

The development of the project was planned and followed through with the use of a Google Sheets document in the form of a Kanban board. The developer had prior experience using this system having previously worked as a web programmer for a startup company, where he was able to see how such methodologies are implemented in a real work environment. This proved to be immensely useful and facilitated the planning of the project.

Chapter 2

PROJECT STRUCTURE

The project can be subdivided into the following parts:

- LAMP server (Linux, Apache, MySQL, PHP)
  - Database
  - ARC Control Panel
  - Skills
  - ARC-Bot
- ARC-Notifications

Figure 3 shows the information flow within the ARC project:

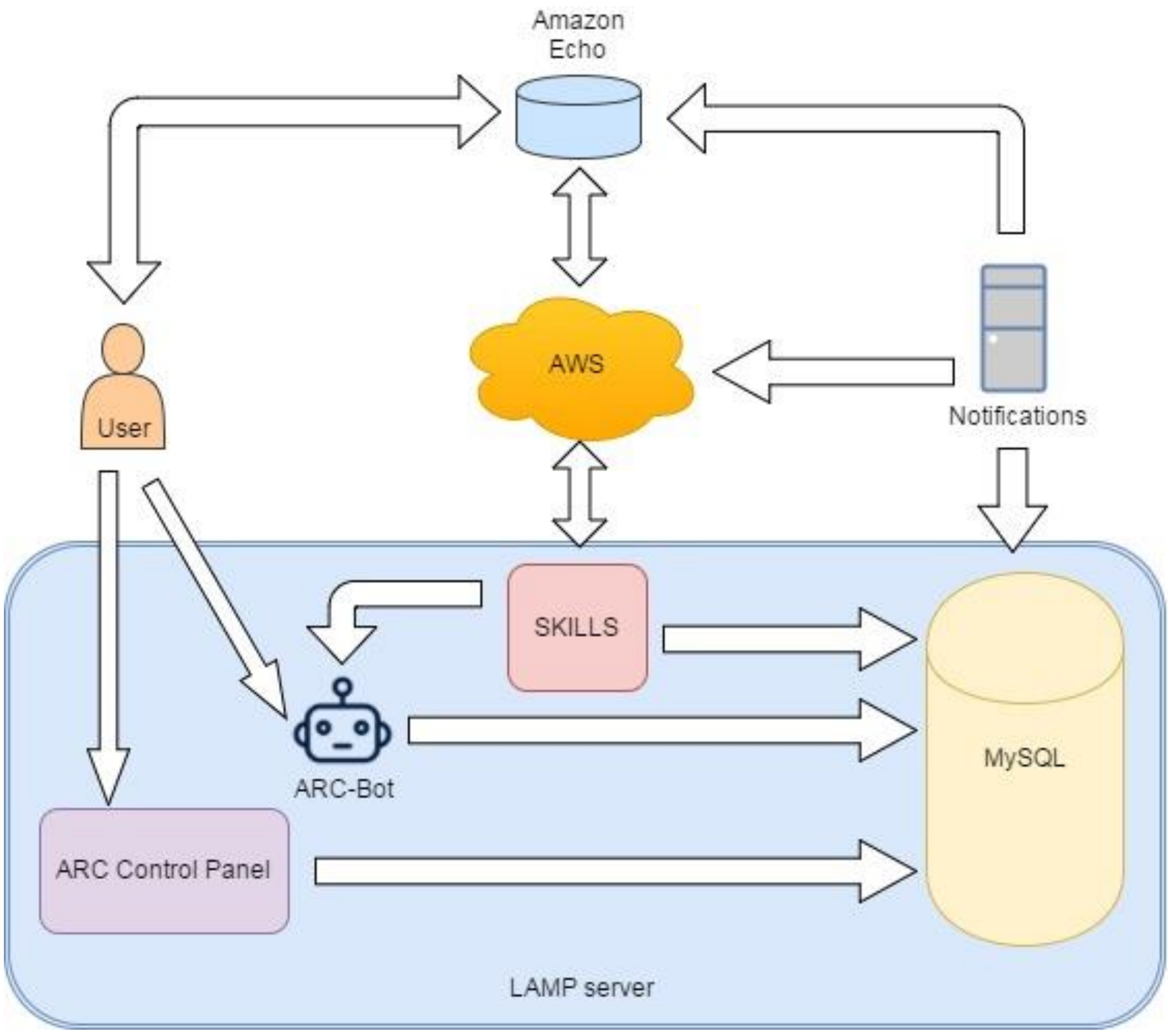


Figure 3: Project information flow

2.1 LAMP server

The LAMP server was built and configured from scratch to feature the following services:

Database

The database contains the user settings for all ARC skills, as well as general ARC settings.

ARC Control Panel

The ARC Control Panel is a website that allows a user to login, change their ARC settings and view their ARC interaction logs.

Skills

The whole ARC project is centered around these skills, giving residential care services to the elderly through an Amazon Echo device.

ARC-Emergency

This skill allows the user to send an emergency message using multiple messaging services to a select group of contacts.

ARC-Medication

This skill acts as a medication reminder and doses explainer in conjunction with a Google Calendar in the form of a medication scheduler.

ARC-Chatbot

This skill acts as an AI conversational module, in conjunction with an HTML-based chatbot.

ARC-Telegram

This skill allows the user to dictate a message and send it to a preprogrammed Telegram chat.

ARC-Bot

This Telegram Bot is the mediator between some of the skills and a Telegram chat of the user’s choosing.

2.2 Notifications

This part of the project works in conjunction with ARC-Medication to add push-notifications to the Amazon Echo, which at the time of this writing lacks this functionality.

Chapter 3

PROJECT DETAIL

This section attempts to explain each part of the project.

3.1 LAMP Server (Linux, Apache, MySQL, PHP)

3.1.1 IntroductionError! Bookmark not defined.

Early on in development the Universitat de Barcelona granted us a server that we converted to a LAMP[4] server (Linux, Apache, MYSQL, PHP). This was done to accommodate the need for a database and a website that uses PHP. It also provided a good platform for hosting our skills.

The software and versions we used to set up a LAMP environment were the following:

- **Debian 8.8**
- **Apache 2.4.10**
- **MYSQL 14.4**
- **PHP 5.6.30**

We were able to control the server remotely using SSH.

3.1.2 Python modules

In this section we will introduce the python modules and components that we used across most of our skills in a similar or equal manner.

httplib2 (version 0.10.3)[5]: a comprehensive HTTP client library.

The class that we used is:

- **Http –** class that represents a client HTTP interface.

This library was used in:

ARC-Emergency
ARC-Medication

ARC-Chatbot
ARC-Telegram
ARC-Notifications

os (version 2.7)[6]: module that provides a portable way of using operating system dependent functionality.

The top-level function that we used is:

- makedirs **–** recursive directory creation function.

The class that we used is:

- **path -** module that implements some useful functions on pathnames. The functions that we used are:
  - expanduser **–** converts the tilde character to a user's home directory.
  - join **–** join one or more path components intelligently.
  - exists **–** returns *True* or *False* depending on if *path* parameter refers to an existing path.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram
ARC-Notifications

apiclient.discovery (version 1.0.2)[7]: a client library for Google's discovery based APIs.

The top-level function that we used is:

- build **–** construct a Resource object for interacting with an API.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram
ARC-Notifications

oauth2client (version 4.1.1)[8]: client library for accessing resources protected by OAuth 2.0.

OAuth 2.0[9]: industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones and living room devices. It is the authorization protocol used by Google APIs.

The classes that we used are:

- **client** – an OAuth 2.0 client. The function that we used is:
  - flow_from_clientsecrets **–** create a Flow from a clientsecrets file.
- **tools** – command-line tools for authenticating. The functions that we used are:
  - argparser **–** ArgumentParser that contains command-line options expected by tools.run().
  - run **–** runs through all the steps to obtain credentials.
  - run_flow **–** takes a Flow argument and attempts to open an authorization server page in the user's default web browser.
- **file –** utilities for making it easier to work with OAuth 2.0 credentials. The class we used is:
  - **Storage –** base class for all Storage objects. Store and retrieve a single credential.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram
ARC-Notifications

flask (version 0.12): microframework implementation of the web frameworks[10] concept.

The class that we used is:

- **Flask –** an instance of this class would be our WSGI application.

This library was used in:

ARC-Emergency
ARC-Medication

ARC-Chatbot
ARC-Telegram

flask_ask (version 0.9.3): easy Alexa Skills Kit integration for Flask.

The classes that we used are:

- **Ask –** an Ask object maps Alexa Requests to flask view functions and handles Alexa sessions. These are the functions of this object that we used:
  - intent – decorator routes an Alexa *IntentRequest* and provides the slot parameters to the wrapped function.
  - launch – decorator maps a view function as the endpoint for an Alexa *LaunchRequest* and starts the skill.
- **statement**[11] **–** one of two classes in Flask-Ask for creating responses.

  Statements terminate Echo sessions. The user is free to start another session, but Alexa will have no memory of it (unless persistence is programmed separately on the server with a database or the like).
- **question** – second class for creating responses. Prompts the user for additional speech and keeps a session open.

  Theses sessions are similar to HTTP sessions but they differ because of a lack of cookies or local storage since the application connects to an Alexa service instead of a browser.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram

flask_sslify (version 0.1.5)[12]: Flask extension that configures a Flask application to redirect all incoming requests to https.

The class that we used is:

- **SSLify** - secures a Flask application.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram

ssl (version 1.16)[13]**:** SSL wrapper for socket objects.

The classes and constants we used are:

- **SSLContext** – instance of an SSL context.
- **PROTOCOL_TLSv1_2** – channel encryption protocol constant.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram

MySQLdb (version 1.16)[14]**:** thread-compatible interface to the popular MySQL database server that provides the Python DB API.

The top-level function that we used is:

- connect **–** creates a connection to the database, returns a Connection object.

The class we used is:

- Cursor – cursor class emulated in MySQLdb. MySQL does not have cursors. The functions that we used are:
  - cursor – creates a cursor from a connection instance.
  - execute – executes an SQL query.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram
ARC-Notifications

ARC-Bot

python-gettext (version 3.0)[15]: Python Gettext *.po* to *.mo* file compiler.

The top-level functions that we used are:

- <u>translation</u> – loads a translation from a local .mo file.
- <u>install</u> – translates the application's strings.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Telegram
ARC-Notifications
ARC-Bot

datetime (version 2.3)[16]: includes functions and classes for doing date and time parsing, formatting, and arithmetic.

The class that we used is:

- **datetime** – A combination of a date and a time. Attributes: year, month, day, hour, monute, second, microsecond and tzinfo. The functions that we used are:
  - <u>now</u> – return the local date and time.
  - <u>date</u> – return a date object with same year, month and day as the datetime object use as a base.
  - <u>time</u> – return a time object with same hour, minute, second and microsecond.
  - <u>utcnow</u> – return the current UTC date and time, with tzinfo None.
  - <u>datetime</u> – class constructor.
  - <u>strftime</u> – return a string representing the date and time, controlled by an explicit format string.

This library was used in:

ARC-Emergency
ARC-Medication
ARC-Telegram
ARC-Chatbot
ARC-Notifications

requests (version 2.16.2)[17]: allows sending of HTTP/1.1 requests.

The class that we used is:

- **Session** – which allows persistence of certain parameters across requests. The functions we used are:
    - session – class constructor.
    - get – performs a get request.
    - post – performs a post request.

This library was used in:

ARC-Emergency
ARC-Telegram
ARC-Chatbot

3.1.3 MYSQL

3.1.3.1 Introduction

In order for the skills to work properly the user has to set up a series of programmable parameters.

Initially, a simple JSON file, housed in the same LAMP server that was hosting the skills, was being used to meet this need and it worked well enough for some time. However, it proved to be insufficient since it required the user to have access to the JSON file and edit in all their parameters, which was tedious at best. Thus, the decision was made to develop a website where the user could log in and customize these parameters to fit their particular needs. These settings are kept in a database within the same LAMP server.

3.1.3.2 ER model and data model

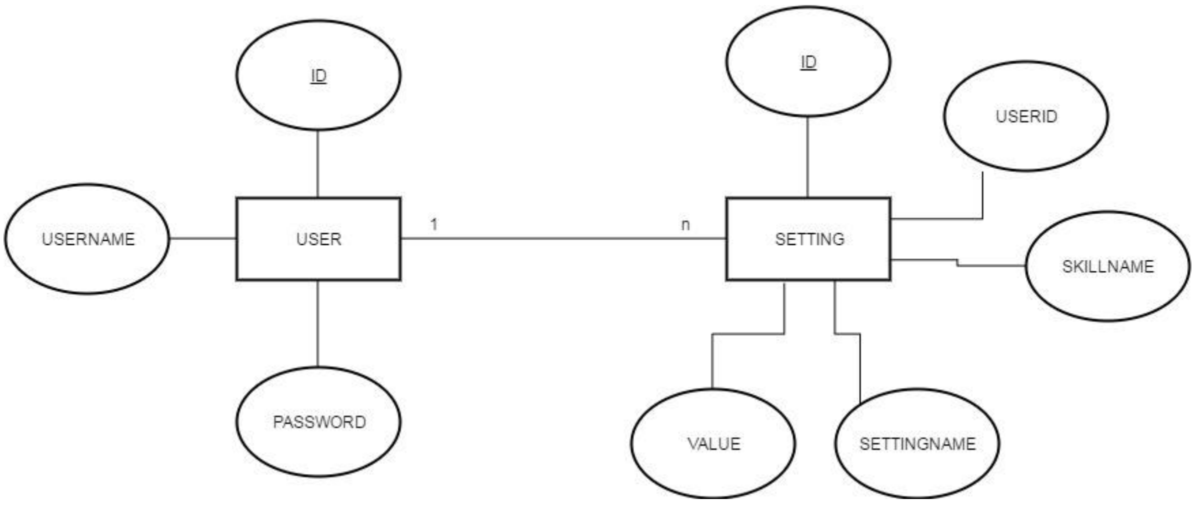
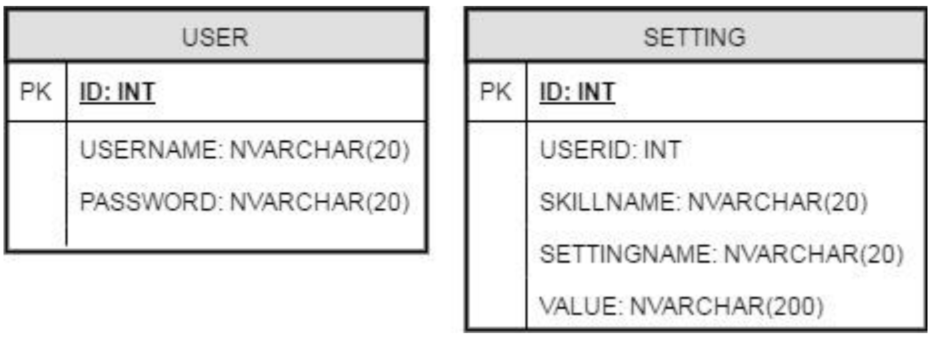


Figure 4: Database ER model



USER	
PK	ID: INT
	USERNAME: NVARCHAR(20)
	PASSWORD: NVARCHAR(20)

SETTING	
PK	ID: INT
	USERID: INT
	SKILLNAME: NVARCHAR(20)
	SETTINGNAME: NVARCHAR(20)
	VALUE: NVARCHAR(200)

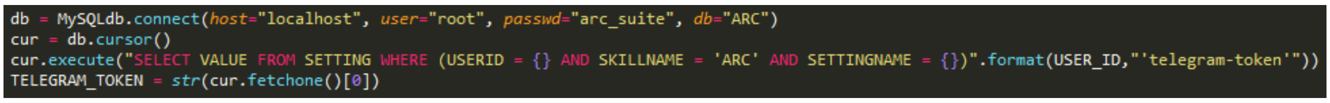
Figure 5: Database data model

There are several reasons for the layout described in the above figures. First of all, two tables make it simpler to program CRUD (Create, Read, Update, Delete) operations. Second, whether it's on the website or while using a skill, we really only need to know the **value** of a setting, which is identified with the USERID, the SKILLNAME and the SETTINGNAME. Therefore, it makes perfect sense to reduce that to a single table, SETTING.

The password, contrary to good practice, is stored in plain text. This decision was made because our project was not intended to be production-ready at the time of handing it in, it was meant to be a proof of concept for the first phase of a larger project still in early development. Therefore, we decided not to investigate hashing passwords for the time being and instead invested our time in more valuable developments.

3.1.3.3 Usage across project parts

The skills, ARC-Notifications and ARC-Bot all read parameters from our database in the following manner:



```
db = MySQLdb.connect(host="localhost", user="root", passwd="arc_suite", db="ARC")
cur = db.cursor()
cur.execute("SELECT VALUE FROM SETTING WHERE (USERID = {} AND SKILLNAME = 'ARC' AND SETTINGNAME = {})".format(USER_ID,"'telegram-token'"))
TELEGRAM_TOKEN = str(cur.fetchone()[0])
```

Figure 6: Reading settings from the database

Using MySQLdb we create a connection to *localhost* specifying the mysql user, password and database name.

Then turn that connection into a cursor and execute an SQL query to retrieve our setting value from the SETTING table, specifying the user ID, skill name and setting name.

3.1.4 ARC Control Panel

3.1.4.1 Introduction

The website which can be accessed using the server's url[18] was made to be a user friendly solution to changing ARC parameters on the fly. We thought it would be a logical solution to the problem of having multiple users with multiple settings each.

3.1.4.2 Objectives

The website should allow the user to:

- Create a user
- Log into their account
- Configure and save parameters (all skills, as well as general settings)
- View interaction logs

3.1.4.3 Technologies used

- PHP
- HTML CSS Bootstrap

3.1.4.4 Web Structure

The website is structured in the following manner:

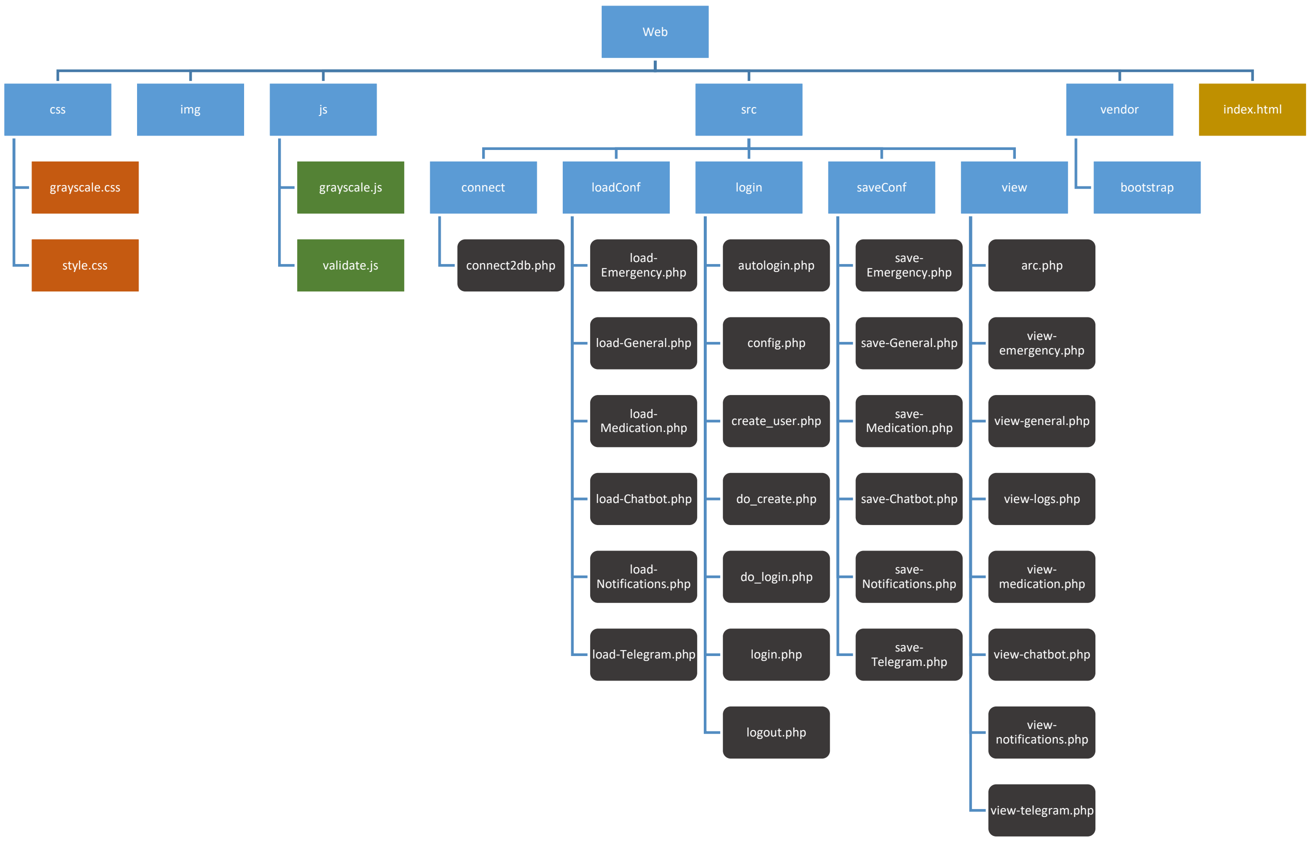


Figure 7: ARC Control Panel file structure

- *css*: contains the css styles for the html, including the Bootstrap theme's css and our own.
- *img*: contains the images used in the site.
- *js*: contains the javascript used in the site, including the Bootstrap theme's javascript and our own.
- *src*:
  - *connect2db*: contains php code to connect to the database. It is included in multiple scripts within the site to avoid repeating code.
  - *loadConf*: contains the php files that load the configurations from the database into the skill configuration pages so that the user can view and edit them.
  - *saveConf*: contains the php files that save the configurations to the database once the user submits their changes.
  - *login*: contains the entire login and create user system.

  - *view*: contains the views. Those would be the main page, arc.php, and all the skill configurations pages (one for each skill as well as a general settings page) plus the local logs view.
- *vendor*: bootstrap files.
- *index.html*: redirects automatically to the main page, arc.php

3.1.4.5 Development

The first requirement was to have a database where we could store this users' account information (username and password).

Once this problem was solved, the second requirement was to have a create/login system.

3.1.4.5.1 Login/Create User

Thankfully we had such a thing in hand, having done *Computació Orientada a la Web* at Universitat de Barcelona in 2015. Coursing this subject, we learned how to develop a fully functioning web-app, including how to set up a login system. We took that system and implemented it fairly easily into our website. This is how it works in our project:

- *arc.php*: main page, if there is no session it redirects the page to login.php
- *login.php*: contains a login form (user, password). If there is already a session it redirects to arc.php. If the form is submitted it switches to do_login.php. This page also links to create.php.
- *do_login.php*: logs the user in if he exists in the database and redirects to arc.php. If not, it redirects to login.php where a login error is displayed. It logs the user by creating a session and storing 3 values: the username and the userID, both from the database, and a Boolean variable used for displaying information in arc.php).
- *config.php*: starts a session.
- *autologin.php*: if the cookie exists, looks for the user in the database and assigns his username to the session as well as his userID.
- *create_user.php*: contains a user creating form (user, password). If submitted, sends the form data to do_create.php.
- *do_create.php*: goes into the database and grabs the largest userID and settingID. It then does the following:

    - Increases the userID by 1 and creates a new user in the database with the submitted information.
    - Adds new entries to the SETTING table, with their default value and assigned to the newly created userID.
- *logout.php*: empties the session and cleans the cookies before redirecting to login.php

Once the login system was set up and running we moved over to building the views.

3.1.4.5.2 Views

Each file named view-{skillname}.php is accompanied by two other files that do the following:

- *view-{skillname}.php*: contains the settings form corresponding to the skill in question. When submitted it sends the information to *save-{skillname}.php.*
- *loadConf/load-{skillname}.php*: this code is included via embedding of php code in *view-{skillname}.php*. It fills the form with the settings in the database corresponding to the particular userID and skill name.
- *saveConf/save-{skillname}.php*: takes the form data from *view-{skillname}.php* and updates the user's settings in the database.

We will go over every user setting once we dive into each skill. However, the user can also change general ARC settings in *view-General.php*.

The general ARC settings that the user can edit are the following:

- Language: English US (EN) or German (DE).
- Log events: true or false. Log the user interactions when using ARC skills.
- Logs location: store logs locally or in a Google Calendar.
- Google Calendar ID: the ID of the Google Calendar used to log the interactions.
- Telegram bot token: identifier of ARC bot for Telegram.

3.1.4.5.3 Bootstrap

Once we had a functioning website we decided to spice things up with a nice Bootstrap theme. We first encountered this front-end framework while coursing Factors Humans at

Universitat de Barcelona. This experience allowed us to choose a theme from startbootstrap.com[19] and quickly apply it to our site.

The theme required little modification. We added a dropdown menu that displays the username and gives the user the option to log out. We also added another dropdown menu that lets the user view both logs.

Other than that, small cosmetic changes were made to the theme that made it fit our site like a glove.

Finally, we made *view-logs.php* which cleanly displays the local logs on a table.

3.1.5 Skills

3.1.5.1 AWS

Skills are uploaded to the AWS through Amazon’s Developer Portal. There a developer can add their own skills and configure them so that they work as intended.

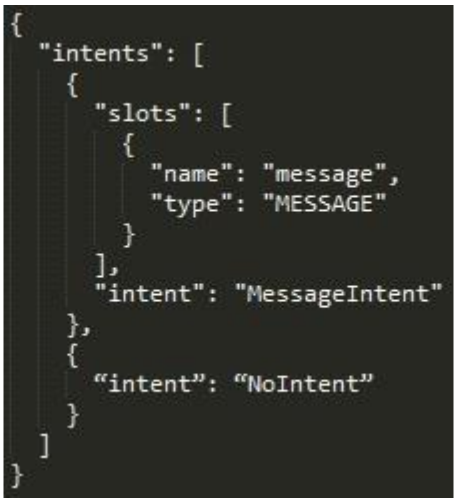
These are the important settings that need to be set up for each skill:

- **Name**: the name of the skill that is displayed to customers in the Alexa app, where they can download any skill available in the catalogue.
- **Invocation name**: the word used by customers to launch a skill. Skills are launched in the following manner:

  “Alexa, launch {invocation name}”

  Instead of launch a variety of words can be used, such as use, start, begin and a bunch of others.

- **Intent Schema**: the schema of user intents in JSON format.

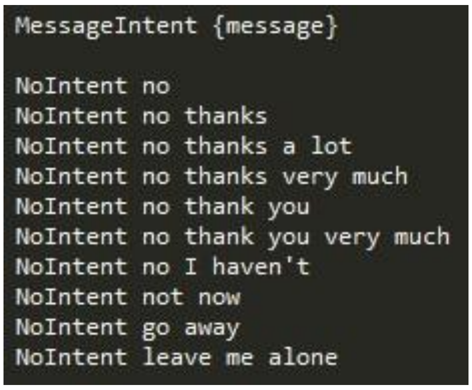


```
{
  "intents": [
    {
      "slots": [
        {
          "name": "message",
          "type": "MESSAGE"
        }
      ],
      "intent": "MessageIntent"
    },
    {
      “intent”: “NoIntent”
    }
  ]
}
```

Figure 8: Example Intent Schema

On figure 8 we can observe the intent schema for a skill that uses two intents and a slot for one of them. Each intent can hold several slots.

- **Custom Slot Types (Optional):** referenced by the Intent Schema and Sample Utterances. A custom slot type defines a list of representative values for the slot. Custom slot types are used for lists of items that are not covered by Amazon’s built-in set of types[20] and are recommended for most use cases. When using a custom type, the developer defines the type and its values.
- **Sample Utterances**: these are sample phrases that the user might utter whilst using the skill. The larger the sample, the bigger the accuracy of Alexa determining the intent that the utterance belongs to.



```
MessageIntent {message}

NoIntent no
NoIntent no thanks
NoIntent no thanks a lot
NoIntent no thanks very much
NoIntent no thank you
NoIntent no thank you very much
NoIntent no I haven't
NoIntent not now
NoIntent go away
NoIntent leave me alone
```

Figure 9:Example Sample Utterances

Figure 9 shows a number of utterances grouped into two different intents.

- **Endpoint**: here the developer can configure the server where the skill is being hosted. There are two options:
    - AWS Lambda: AWS Lambda is a server-less compute service that runs code in response to events and automatically manages the underlying resources. This is the recommended option for a production-ready skill.
    - HTTPS: alternatively, one can host their skill on a server of their choosing, so long as HTTPS is used for requests. This is the method we used.
- **Certificate for endpoint**: to protect the security of end users Amazon requires that the developer uses a certificate while developing an Alexa skill. They offer three options:
    - *My development endpoint has a certificate from a trusted certificate authority.*
    - *My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority.*
        - *NGrok*[21] allows you to expose a local server behind a NAT or firewall to the internet. Early on in development we used it so that it was possible to start making and testing skills straight away. But it proved to be an uncomfortable method and so we switched to serving https from our skills being run on the LAMP server as soon as it was possible.
    - *I will upload a self-signed certificate in X.509 format.*

3.1.5.2 Https and Certificate:

Because we are serving the skills from a server of our choosing, the requests need to use HTTPS in order for AWS to accept them.

We configured the web server to use a self-signed certificate. Amazon offers an excellent tutorial on how to do this. [22]

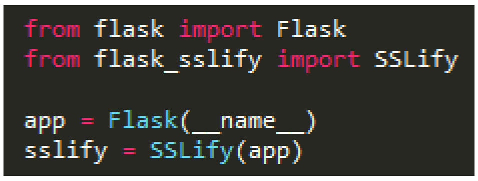
Then we had to update our skills to serve https, which we did using the following python modules:

- **Flask-SSLify**: this is a simple Flask extension that configures your Flask application to redirect all incoming requests to https.
- **ssl**: this module provides access to Transport Layer Security encryption and peer authentication facilities for network sockets, both client-side and server-side.

3.1.5.2.1 Application across skills

Https is applied to our skills by doing two things.

Firstly, we redirect all incoming requests to https using Flask-SSLify. This is done in the following manner:

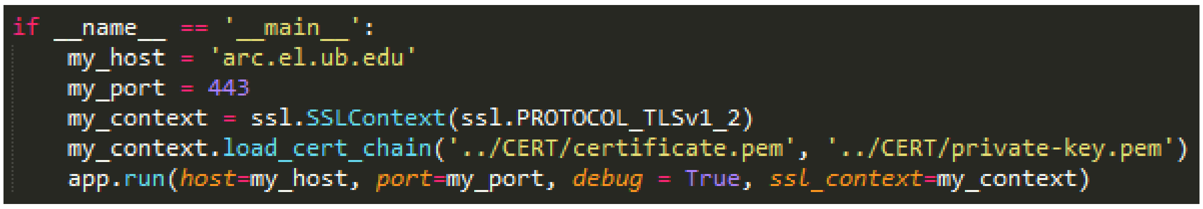


```python
from flask import Flask
from flask_sslify import SSLify

app = Flask(__name__)
sslify = SSLify(app)
```

Figure 10:flask_sslify application on our skills

Secondly, we run our skill like so:



```python
if __name__ == '__main__':
    my_host = 'arc.el.ub.edu'
    my_port = 443
    my_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    my_context.load_cert_chain('../CERT/certificate.pem', '../CERT/private-key.pem')
    app.run(host=my_host, port=my_port, debug = True, ssl_context=my_context)
```

Figure 11: Launching a skill with https

Here we specify the host url, port, SSL context and self-signed certificate.

3.1.5.3 Internationalization:

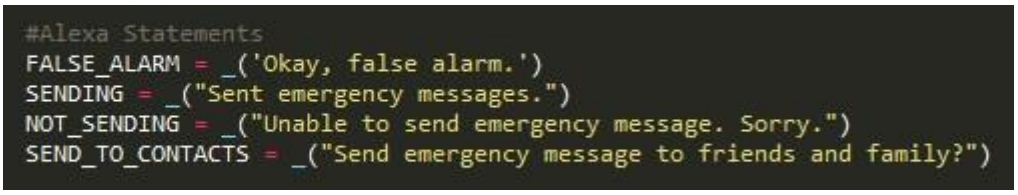
Internationalization was achieved thanks to python's *gettext* module. This simple solution allows for a program to have multiple translations and to be able to run in any language made available by the developer.

We applied internationalization to ARC-Notifications, ARC-Bot and all our skills with the exception of ARC-Chatbot. The reason for this decision is that the chatbot we use currently only speaks English, so internationalizing that skill made little to no sense.

Here is how we applied gettext to our skills.

Step 1: Replacing strings

For every string that Alexa is going to utter, it is necessary to bound the string in a function call such as the one shown in figure 12.



```
#Alexa Statements
FALSE_ALARM = _('Okay, false alarm.')
SENDING = _("Sent emergency messages.")
NOT_SENDING = _("Unable to send emergency message. Sorry.")
SEND_TO_CONTACTS = _("Send emergency message to friends and family?")
```

Figure 12: Strings bound in function call for internationalization

It also helps to group all would-be-spoken strings together, it makes it easier to find and modify them.

Step 2: Extract the strings

With gettext we can extract the strings into a POT file, which is a template file for PO files. This can be done in the following manner:

Python pygettext.py –d arc-emergency arc-emergency.py

Step 3: Translate the strings using Poedit

Poedit is a program that takes .pot files and lets us create new translations for our strings. Once this is done we can save the translations, which generates the following files for each language:

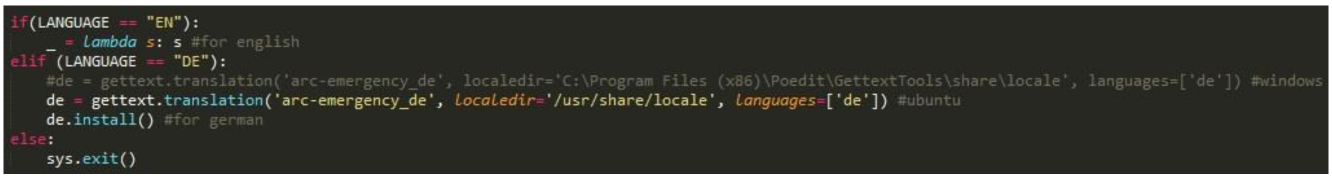
- .po: contains the actual translation.
- .mo: binary data file, used by the GNU gettext program to translate our code. Again, each language has its own MO file.

Step 4: Add gettext code to our program

To make our program work with gettext the first thing we had to add is the import:

Import gettext

Followed by:



```
if(LANGUAGE == "EN"):
    _ = lambda s: s #for english
elif (LANGUAGE == "DE"):
    #de = gettext.translation('arc-emergency_de', localedir='C:\Program Files (x86)\Poedit\GettextTools\share\locale', languages=['de']) #windows
    de = gettext.translation('arc-emergency_de', localedir='/usr/share/locale', languages=['de']) #ubuntu
    de.install() #for german
else:
    sys.exit()
```

Figure 13: Python code used to translate the strings in a skill

Where “LANGUAGE” is a skill setting previously retrieved from the database. This should be executed as early as possible, that way the skill is translated before Alexa starts speaking to the user.

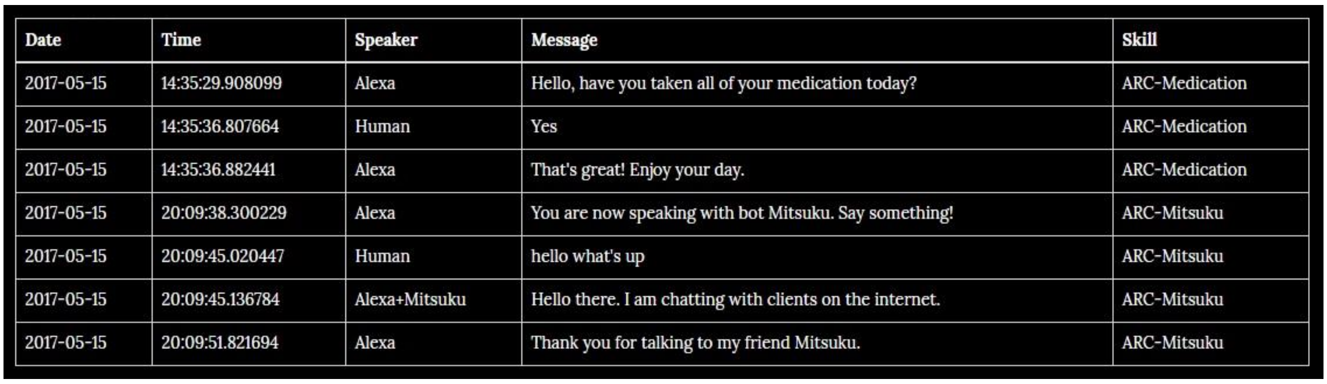
The install() method causes all the _() calls to return the translated string. If the original translation is needed however we can simply assign a lambda function value to _ that returns the string it was passed, as seen in figure 13.

3.1.5.4 Logs

There are two ways to log the interactions between the user and Alexa while using an ARC skill. During testing of the Google Gmail API we considered the possibility of making a log system, so we quickly learned to use the Calendar API and put it to use. It worked, but it was slow. We approached the problem again a few months later once most of the skills had been developed, our solution was to log the interactions to a JSON file in the same server that was running the skills. This fixed the speed issue, but we still liked viewing the logs in a calendar display, so we decided to keep both options.

As a result, the user can save logs if they please. And if that’s the case then they can choose one of the two given options.

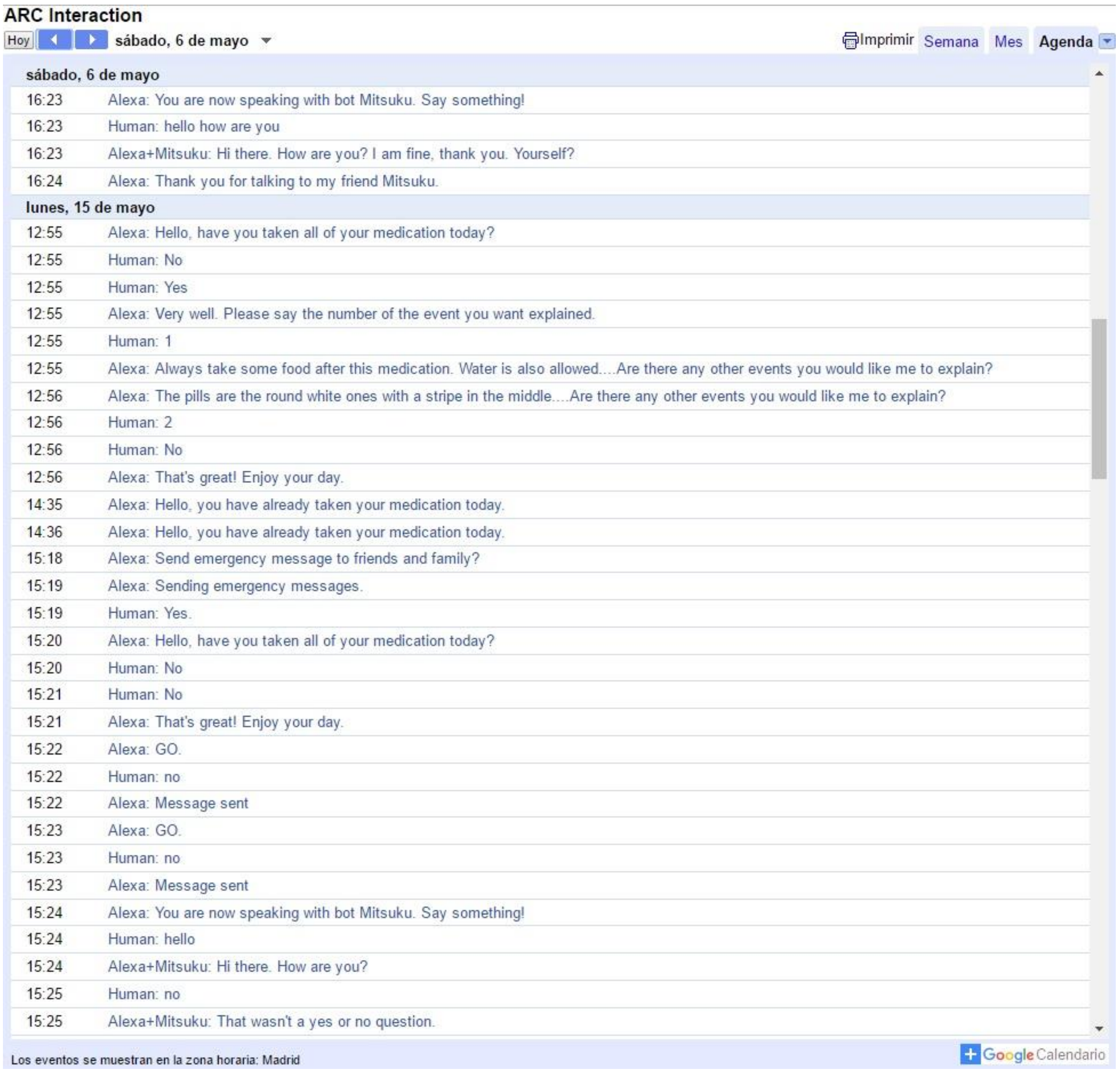
Local logs look like this (as seen in the ARC Control Panel):



Date	Time	Speaker	Message	Skill
2017-05-15	14:35:29.908099	Alexa	Hello, have you taken all of your medication today?	ARC-Medication
2017-05-15	14:35:36.807664	Human	Yes	ARC-Medication
2017-05-15	14:35:36.882441	Alexa	That's great! Enjoy your day.	ARC-Medication
2017-05-15	20:09:38.300229	Alexa	You are now speaking with bot Mitsuku. Say something!	ARC-Mitsuku
2017-05-15	20:09:45.020447	Human	hello what's up	ARC-Mitsuku
2017-05-15	20:09:45.136784	Alexa+Mitsuku	Hello there. I am chatting with clients on the internet.	ARC-Mitsuku
2017-05-15	20:09:51.821694	Alexa	Thank you for talking to my friend Mitsuku.	ARC-Mitsuku

Figure 14: ARC Local Logs

Meanwhile calendar logs look like this:



```
sábado, 6 de mayo
16:23  Alexa: You are now speaking with bot Mitsuku. Say something!
16:23  Human: hello how are you
16:23  Alexa+Mitsuku: Hi there. How are you? I am fine, thank you. Yourself?
16:24  Alexa: Thank you for talking to my friend Mitsuku.
lunes, 15 de mayo
12:55  Alexa: Hello, have you taken all of your medication today?
12:55  Human: No
12:55  Human: Yes
12:55  Alexa: Very well. Please say the number of the event you want explained.
12:55  Human: 1
12:55  Alexa: Always take some food after this medication. Water is also allowed....Are there any other events you would like me to explain?
12:56  Alexa: The pills are the round white ones with a stripe in the middle....Are there any other events you would like me to explain?
12:56  Human: 2
12:56  Human: No
12:56  Alexa: That's great! Enjoy your day.
14:35  Alexa: Hello, you have already taken your medication today.
14:36  Alexa: Hello, you have already taken your medication today.
15:18  Alexa: Send emergency message to friends and family?
15:19  Alexa: Sending emergency messages.
15:19  Human: Yes.
15:20  Alexa: Hello, have you taken all of your medication today?
15:20  Human: No
15:21  Human: No
15:21  Alexa: That's great! Enjoy your day.
15:22  Alexa: GO.
15:22  Human: no
15:22  Alexa: Message sent
15:23  Alexa: GO.
15:23  Human: no
15:23  Alexa: Message sent
15:24  Alexa: You are now speaking with bot Mitsuku. Say something!
15:24  Human: hello
15:24  Alexa+Mitsuku: Hi there. How are you?
15:25  Human: no
15:25  Alexa+Mitsuku: That wasn't a yes or no question.
```

Figure 15: ARC Calendar Logs

3.5.4.1 Application across our project

Log settings are read, along with other settings, from the database. The log settings are:

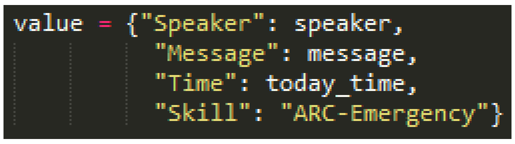
- Log events: Boolean that indicates if the application is to log events or not.
- Logs location: local or using a google calendar
- Log calendar ID: used to identify the Google Calendar used to log the interactions.

The value of the log events Boolean is checked every time either Alexa or the user say something. In case of being True, a logging method is called. There are two options, depending on the logs location setting:

- log_event_calendar: uses the quickAdd method from the Google Calendar API to add an event to the specified calendar. Because we use it as a log system, the event name has the following format:

  <person>: <phrase>

  Where person can either be Alexa or Human.
- log_event_local: opens the json log file corresponding with the user's ID. It then adds an entry with today's date as the key and the following structure as the value:



```
value = {"Speaker": speaker,
         "Message": message,
         "Time": today_time,
         "Skill": "ARC-Emergency"}
```

Figure 16: Local log entry structure.

When using a Google API, one needs to define the scopes. This indicates to the API the functions that the application is authorized to execute.

In order to log events into a Google Calendar the following scope is needed:

https://www.googleapis.com/auth/calendar

3.1.5.5 ARC-Emergency

3.1.5.5.1 Introduction

This skill was developed in the hopes of providing an easy way for Echo users to send a predetermined emergency message to select contacts with little to no effort.

3.1.5.5.2 Objectives

The objectives for building this skill were the following:

- Require simple dialog with the user so as to send the message as quickly as possible.
- Use multiple messaging services, lessening the probability that the message won't be delivered due to external causes.

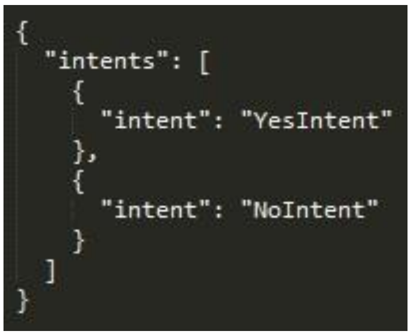
3.1.5.5.3 Prerequisites

In order for this skill to work as intended some steps need to be taken beforehand.

<u>AWS configuration</u>

This configuration is made by the skill's developer.

- **Skill name**: ARC Emergency
- **Invocation name**: emergency
- **Intent schema**: the intent schema used for this skill is the following:



```
{
  "intents": [
    {
      "intent": "YesIntent"
    },
    {
      "intent": "NoIntent"
    }
  ]
}
```

Figure 17: ARC-Emergency intent schema

- **Sample Utterances**:

YesIntent yes
YesIntent yes please
YesIntent do it
YesIntent sure
YesIntent of course
YesIntent obviously
YesIntent go
NoIntent no
NoIntent no sorry
NoIntent not now
NoIntent go away
NoIntent leave me alone
NoIntent nevermind
NoIntent false alarm
NoIntent nevermind Alexa
NoIntent nevermind Computer

- **Endpoint**:

  https://arc.el.ub.edu:443/arc_emergency

ARC Control Panel Settings

In the ARC Control Panel, the user can configure their settings for this skill. The settings are the following:

General

- Enabled messaging services: The messaging services that the skill will use to send an emergency message. The user has three options to choose from:
  - Gmail
  - Telegram
  - Gmail and Telegram

Gmail

- From: the email account from which the messages will be sent.
- To: the recipients, separated by commas.
- Subject: the email subject field.
- Message: the email text (the message).

Telegram

- Chat_ID: identifies the telegram chat that will receive the message.
- Message: the Telegram message.

ARC Bot

In order to use the Telegram messaging service with this skill, a specific telegram bot needs to be running.

The user also needs to create a group chat in the Telegram app and invite the recipient contacts, as well as the telegram bot.

Finally, the telegram bot needs to have admin privileges within the group chat.

ARC bot was created by us for this specific purpose and will be expanded on later.

3.1.5.5.4 Development

Python modules used

- **apiclient->errors**[23]**:** errors for the apiclient library.

  We used the following exception when sending an email using the Google Gmail API:

  - HttpError – error class for unexpected or invalid HTTP data.

- **email.mime.text->MIMEText**[24]: class used to create MIME objects of major type *text*.
  We used the class constructor to create a MIMEText from our email message string, as required by the Google GmailAPI.

- **base64**[25]: module that provides data encoding and decoding as specified in RFC 3548[26]. We use the following method:
  - b64encode – to encode our emails using the base64 algorithm.

The application's execution is segregated according to the following decorators:

- *@ask.launch* – this decorator maps a view function as the endpoint for an Alexa LaunchRequest and starts the skill. This function, in this case, reads the parameters from the database and returns a welcome message in the form of a question, asking the user if they want to send an emergency message to their contacts.
- *@ask.intent("YesIntent")* – this decorator maps the YesIntent to a function that calls other functions charged with sending the emergency message in the various ways available, then later informs the user if the action was a success or not.
- *@ask.intent("NoIntent")* – this decorator maps the NoIntent to a function that returns a statement acknowledging the user's change of mind regarding sending the messages.

Telegram messages are sent by sending a POST request to:

```
https://api.telegram.org/bot{bot_token}/sendMessage?text={message}&chat_id={telegram_chat_id}
```

Emails are sent using the Google Gmail API using the following instruction:

```
service.users().messages().send(userId=user_id, body=message).execute()
```

The Google API scope needed to create and send emails is:

```
https://www.googleapis.com/auth/gmail.compose
```

3.1.5.6 ARC-Medication

3.1.5.6.1 Introduction

This skill is the most extensive of the suite. Its purpose is to act as a medication reminder and doses explainer, in conjunction with a medication scheduler.

3.1.5.6.2 Objectives

The objectives for building this skill were the following:

- Use or create a user-friendly way to input the medication, times, and explanations.
- Avoid making an intrusive or annoying skill. This could be seen as a general rule for developing any skill, but this one in particular had to have several branches of dialogue so it was of particular importance to keep the principle in mind during design and development.

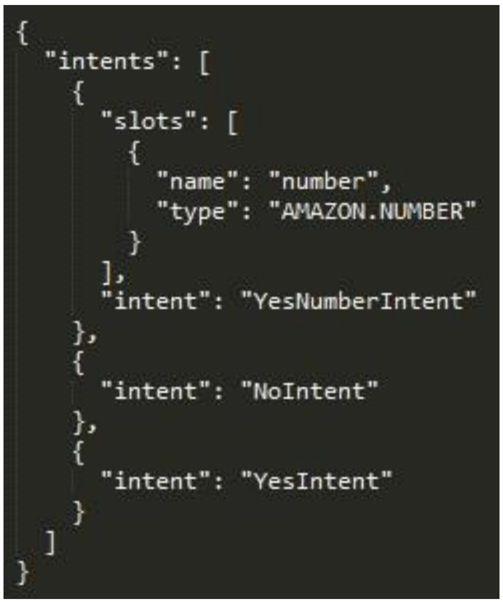
3.1.5.6.3 Prerequisites

In order for this skill to work as intended some steps need to be taken beforehand.

<u>AWS configuration</u>

This setup is performed by the developer, not the skill user.

- **Skill name**: ARC Medication
- **Invocation name**: medication
- **Intent schema**: the intent schema used for this skill is the following:



```
{
  "intents": [
    {
      "slots": [
        {
          "name": "number",
          "type": "AMAZON.NUMBER"
        }
      ],
      "intent": "YesNumberIntent"
    },
    {
      "intent": "NoIntent"
    },
    {
      "intent": "YesIntent"
    }
  ]
}
```

Figure 18: ARC-Medication intent schema

- **Sample Utterances**:

YesNumberIntent yes {number}

YesNumberIntent yes please explain number {number}

YesNumberIntent number {number} please

YesNumberIntent number {number} thank you

YesNumberIntent number {number} thank you very much

YesNumberIntent yes {number} please

YesNumberIntent yes number {number} please

YesNumberIntent number {number}

NoIntent no

NoIntent no thanks

NoIntent no thanks a lot

NoIntent no thanks very much

NoIntent no thank you

NoIntent no thank you very much

NoIntent no I haven't

NoIntent not now

NoIntent go away

NoIntent leave me alone

YesIntent yes

YesIntent yeah

YesIntent yes I have

YesIntent yes thank you for asking

YesIntent sure

YesIntent of course

YesIntent obviously

- **Endpoint**:

  https://arc.el.ub.edu:443/arc_medication

Google Calendar

This skill is linked to a Google Calendar that belongs to the user and acts as a medication scheduler. This is done by creating an event for every time that the patient must take medication. This is what needs to be specified for each event:

- Event title: dose of medication (e.g.“1 red pill” or “2 white pills with a stripe”).
- Start time: the time at which the patient has to take this medication.
- Description: explanation for the dose or instructions (e.g. “Always take some food after this medication” or “Take with water”).

ARC Control Panel Settings

In the ARC Control Panel the user can configure their settings for this skill. The settings are the following:

General

- Calendar ID: identifies the Google Calendar containing the medication.
- Person name: Alexa refers to the user by this name.
- Last check: the last date that this skill was executed.
- Taken meds today: two options, true or false.

3.1.5.6.4 Development

Python modules used

- **dateutil (version 2.6.0)**[27]: module that provides powerful extensions to the datetime module available in the Python library.

The application's execution is segregated according to the following decorators:

- @ask.launch – when the application is launched, the skill parameters are read from the database. ARC-Medication has two parameters that help the application figure out if the patient needs medication information or not. Those two parameters are:
    - *takenMeds-date*: the last date that the application was run. We assume that the patient took their medication if they launched the skill, therefore this is representative of the last date that the patient took their medication. The format for the date is "year-month-day".
    - *takenMeds-bool*: datatype Boolean. Indicates whether the patient took their medication today or not. This variable is only used if the value *takenMeds-date* is today's date.

  When the parameters are read, the value of these two settings are looked at before continuing execution of the app. If *takenMeds-date* has a value equal to today, then *takenMeds-bool* can be seen as a reliable source of information. If the opposite is true, then *takenMeds-bool* is set to False and *takenMeds-date* is set to today's date. These changes are performed both locally and in the database, as soon as the values are known to the application.

  Once these operations have ocurred, two things will happen depending on the value of *takenMeds-bool*:
    - If True, this means that the patient has already taken their medication today and therefore does not need any instructions. Alexa returns a **statement** informing the patient as such.
    - If False, this means that the patient has NOT taken their medication today, and therefore might require information. However, because we don't want our app to be overly intrusive, Alexa returns a **question** asking the patient if they have taken their medication today or not.

Before diving into the rest of the decorators, we first have to talk about an inherent problem that arises when designing an application with a voice-based interface.

If the application is complex enough, there might be different situations in which the user replies to Alexa with the same response. Here's an example:

> *1-Alexa: are you tired?*
> *2-Human: yes.*
> *3-Alexa: would you like me to dim the lights?*
> *4-Human: yes.*

Because an intent is mapped to a single method using a decorator, state variables are required for the application to know at what point of the conversation tree it is at.

In the case of ARC-Medication, the user can be asked two different questions that require a yes or no answer (YesIntent or NoIntent). We use a boolean variable called *TIME_TO_EXPLAIN* to navigate this. When the app is launched the value of this variable is False, and it is changed to True once Alexa reads the medication events from the calendar and asks the user if they would like a dose explanation.

- *@ask.intent("YesIntent")* – If *TIME_TO_EXPLAIN* is False, the user will have answered with a *YesIntent* to the question "Have you taken your medication today?". At this point the *takenMeds* variables are updated in the database and Alexa returns a statement telling the user that they did a good job.
  If, however, *TIME_TO_EXPLAIN* happens to be True then the user will have answered with a *YesIntent* to the question "Would you like me to explain any of those?", referring to the medication events that Alexa will have read out loud before the user's yes intent. In this case, Alexa returns a question asking the user which number of medication event they would like explained.

- *@ask.intent("NoIntent")* – If *TIME_TO_EXPLAIN* is False, the user will have answered with a *NoIntent* to the question "Have you taken your medication today?" In this case, the patient requires medication instructions. Firstly, the *takenMeds* settings are updated in the database (*takenMeds-date -> today, takenMeds-bool -> True*). Then, the medication events are extracted from the calendar and formatted into an intelligible string that Alexa can read back to the user. These events are also stored inside a list, for later use. Lastly, the *TIME_TO_EXPLAIN* variable is changed to True and Alexa reads the instructions to the user, after which it asks them if they need a dose explanation.

If, however, *TIME_TO_EXPLAIN* is False when this method is used, Alexa returns a statement closing the session.

- *@ask.intent("YesNumberIntent")* – this intent is an answer to any of these two questions:

  Alexa: <medication events> + "Would you like me to explain any of those?"

  Alexa: <dose explanation> + "Are there any other events you would like me to explain?"

  In both cases, the user answers with the number of event that they want explained.

  Because the application has previously stored the events in a list, it can check whether this number of event exists or not. If it exists, it takes the event's description and returns it in the form of a question (asking afterwards if they want any other event explained).

  If the number does not exist, however, it informs the user of this issue and asks them to inquire about a different event.

Medication events are taken from the google calendar and formatted in a function called *get_instructions()*.

Using the Google Calendar API the skill calls upon the following method:

```
service.events().list(calendarId, timeMin, timeMax, singleEvents, orderBy)
```

The parameters being the following:

-calendarId -> ID of Google Calendar acting as medication scheduler.

-timeMin -> all events before this time are ignored.

-timeMax -> all events after this time are ignored.

-singleEvents -> whether or not we want repeated events.

-orderBy -> parameters by which the items list is ordered.

We decided that relevant events should be the ones happening on the same day as the inquiry. With that in mind we used *datetime* to get the earliest moment (00:00:00) and the last moment (23:59:59) of the current day and use it as the time filter.

The API method returns a dictionary that contains a list of events, called ‘items’. This list is saved in order to offer dose explanations.

Once the list is saved, the skill goes over every event and prepares a list of instructions as strings. Each string has the following form:

```
‘Take’ + eventSummary + ‘ at ‘ + hour + ‘ ‘ + minute + m
```

With the variables being:

-eventSummary -> Name of the event (e.g. “2 white pills”)

-hour -> number from 0 to 12.

-minute -> number from 0 to 59.

-m -> AM or PM

After the loop the list of instructions is joined into a single string and returned.

3.1.5.7 ARC-Chatbot

3.1.5.7.1 Introduction

Alexa, though it’s an excellent home assistant, lacks the ability to have long coherent conversations. By contrast, chatbots are special AIs that attempt to keep a conversation with a person, providing automatic responses to entries made by the user.

Chatbots are commonly programmed using Artificial Intelligence Markup Language (AIML), an XML-compliant language. It was designed by the creators of A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), one of the most robust programs of its kind, and it makes it possible to customize an Alicebot or to create one from scratch.

Harnessing the potential of skills, our idea was to create a skill where the user would say something to Alexa, who would send that message to a chatbot, get a response and say it back to the person. By speaking to the chatbot through Alexa, the user would be able to have a coherent spoken conversation with an AI using their voice.

3.1.5.7.2 Objectives

The objectives for building this skill were the following:

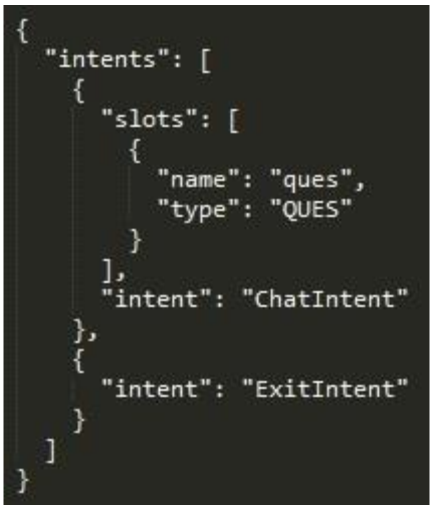
- Achieve uninterrupted dialogue between the user and a chatbot.

3.1.5.7.3 Prerequisites

In order for this skill to work as intended some steps need to be taken beforehand.

<u>AWS configuration</u>

- **Skill name**: ARC Chatbot
- **Invocation name**: chatbot
- **Intent schema:** the intent schema used for this skill is the following:



```
{
  "intents": [
    {
      "slots": [
        {
          "name": "ques",
          "type": "QUES"
        }
      ],
      "intent": "ChatIntent"
    },
    {
      "intent": "ExitIntent"
    }
  ]
}
```

Figure 19: ARC-Chatbot intent schema

- **Custom Slot Types:**

  Type: *QUES*

  Values:

  what is your name
  ioesrs goast julius mandatory whatever
  s dfgdfgdfngjd fgdjfafjilxco
  aoiaeir oirasi delta airlines
  this is a test of skill
  and such and stuff and things and more
  toast is good I think let's ask
  don't do it or else please thank you

Alexa attempts to map a user's utterance with the sample utterances defined by the developer. This is a problem if we can't predict what the user is going to say at all.

Because this skill allows the user to freely talk with a chatbot, this skill suffers from this problem.

Alexa maps the user's utterance to the closest sounding sample utterance, and thus assigns it the corresponding intent.

However, after some investigating we learned that if no match is found, it assigns it to the intent with the largest amount of sample utterances.

Our solution to the problem was to create two different intents, one for saying something to the bot and another to exit the chat. We made a slot for the former, and defined several random values to the slot. After thorough testing this proved to be an excellent solution.

- **Sample Utterances:**

  ChatIntent {ques}

  ExitIntent Bye bye

- **Endpoint:**

  https://arc.el.ub.edu:443/arc_chatbot

ARC Control Panel Settings

In the ARC Control Panel the user can configure their settings for this skill. The settings are the following:

- **General**
  - Chatbot URL: URL of an html-based Chatbot chat.

3.1.5.7.4 Development

- **bs4 – BeautifulSoup (version 4.6.0)**[28]: Python library for pulling data out of HTML and XML files. Works with a variety of parsers to provide idiomatic ways of navigating, searching and modifying the parse tree.

The application's execution is segregated according to the following decorators:

- *@ask.launch* – the first step taken when launching this skill, after reading the skill parameters from the database, is to perform a GET request on the chatbot's URL. If a status code 200 is returned, Alexa informs the user that they are now speaking to the chatbot and asks them, using a Flask-ASK *question*, to say something.
- *@ask.intent("ChatIntent")* – this function has a custom slot as a parameter. This slot is the user's entire utterance and is sent to the chatbot in the form of a POST request. If a status code 200 is returned, then the chatbot's response is extracted using BeautifulSoup.

  The returned chatbot html is converted to a BeautifulSoup instance, using the *lxml* parser:

  ```
  soup = BeautifulSoup(req.text, "lxml")
  ```

  Because the chatbot's response sometimes contains html elements such as links which are irrelevant to a user interacting with a vocal interface (they can't click links or view images), these elements need to be extracted. This is done using the same library:

  ```
  [s.extract() for s in soup('script')]
  [s.extract() for s in soup('bgsound')]
  [s.extract() for s in soup('a')]
  [s.extract() for s in soup('img')]
  ```

  Then, the div with an id equal to "output" is confined:

  ```
  soup = soup.find("div", id = "output").get_text(strip=True)
  ```

  Finally, the remaining text is encoded with utf-8 and read by Alexa using a Flask-ASK question, that way the conversation is on-going.
- *@ask.intent("ExitIntent")* – Alexa reads a closing statement in the form of a Flask-ASK statement, thereby ending the skill's session.

3.1.5.8 ARC-Telegram

3.1.5.8.1 Introduction

After we developed ARC-Emergency we had a sense of how to use skills in conjunction with a telegram bot. We capitalized on this and decided to build an additional skill where the user could speak any message and Alexa would send it to a telegram chat.

3.1.5.8.2 Objectives

The objectives for building this skill were the following:

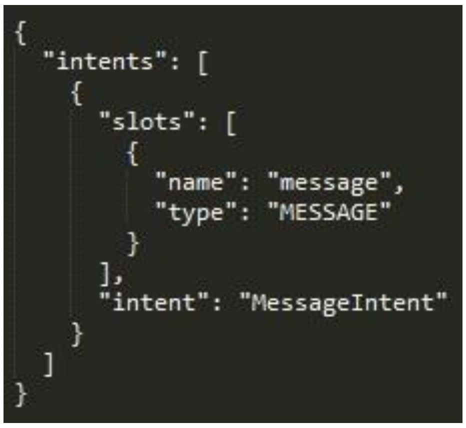
- Send a spoken message as text to a telegram chat.

3.1.5.8.3 Prerequisites

In order for this skill to work as intended some steps need to be taken beforehand.

<u>AWS configuration</u>

- **Skill name**: ARC Telegram
- **Invocation name**: telegram
- **Intent schema**: the intent schema used for this skill is the following:



```
{
  "intents": [
    {
      "slots": [
        {
          "name": "message",
          "type": "MESSAGE"
        }
      ],
      "intent": "MessageIntent"
    }
  ]
}
```

Figure 20: ARC-Telegram intent schema

- **Custom Slot Types**:

  Type: *MESSAGE*

  Values:

  what is your name
  ioesrs goast julius mandatory whatever
  s dfgdfgdfngjd fgdjfafjilxco
  aoiaeir oirasi delta airlines
  this is a test of skill
  and such and stuff and things and more
  toast is good I think let's ask
  don't do it or else please thank you

This skill suffers from the same problem as ARC-Chatbot when it comes to expected utterances from the user, so we adopted the same solution.

- **Sample Utterances**:

  MessageIntent {message}

- **Endpoint:**

  https://arc.el.ub.edu:443/arc_telegram

ARC Control Panel Settings

In the ARC Control Panel the user can configure their settings for this skill. The settings are the following:

- **General**
  - Telegram Chat ID: the chat's ID.

3.1.5.8.4 Development

The application's execution is segregated according to the following decorators:

- *@ask.launch* – as soon as the skill is launched by a user, this function reads the settings from the database and asks the user for a message.
- *@ask.intent("MessageIntent")* – this function uses a custom slot as a parameter, just like ARC-Chatbot, that contains the entire user's utterance. The message is then sent as a POST request to the following url:

https://api.telegram.org/bot{bot_token}/sendMessage?text={message}&chat_id={telegram_chat_id}

Whether the request is sent successfully or not, Alexa informs the user in both cases with a closing statement.

3.1.6 ARC-Bot

3.1.6.1 Introduction

In order to send a message to a telegram chat from a platform that isn't the Telegram app one needs to create and use a telegram bot.

Bots[29] are Telegram accounts operated by software – not people – and they often have AI features. They can do anything – teach, play, search, broadcast, remind, connect, integrate with other services, or even pass commands to the internet of Things.

Bots are third-party applications that run inside Telegram. Users can interact with bots by sending messages, commands, and inline requests.

A telegram bot, therefore, is central to any skill that intends to interact with Telegram.

3.1.6.2 Technologies used

- **telebot (version 0.0.2)**[30]: a Telegram bot library, with simple route decorators.

Why telebot?

Currently there is a plethora of Telegram bot APIs. Telebot, though a work in progress, is simple to use and has enough functionality to suit our purpose.

3.1.6.3 Development

Before writing the bot's code the first step we had to take was to create a bot.

Creating and setting up a bot

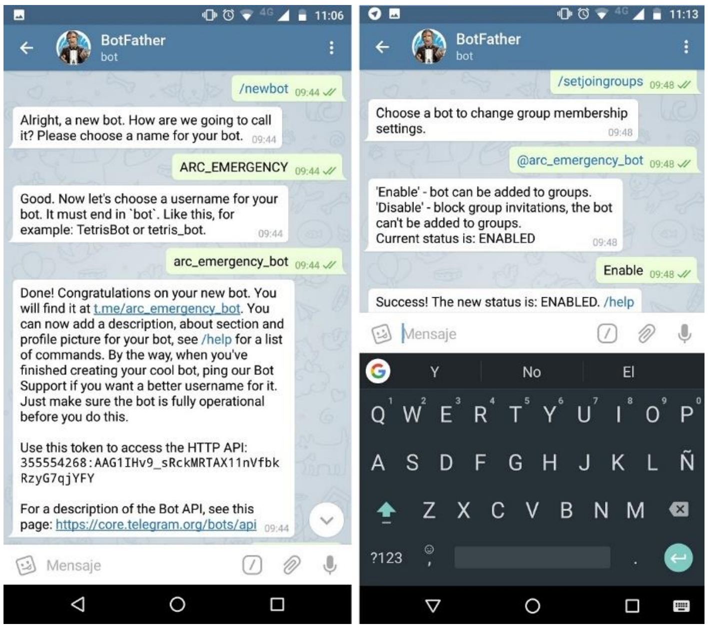
The BotFather[31] is a bot created by the Telegram team that users must use in order to create and manage a bot of their own.

It has a great number of commands but we're really only interested in a few.

- /newbot: the BotFather will follow this command by asking the user for a name by which to call the new bot and a username so that users can find it.
- /setjoingroups: set whether a bot can join groups.
  - Bots can't initiate conversations with users. A user **must** either add them to a group or send them a message first. For our project we needed a bot that

would send messages to a group of people (family, close friends, etc) so we chose the former method.

- /mybots: edit a user's bot. We can change the name, description, about, botpic and commands, as well as obtain the bot's API token.



/newbot 09:44

Alright, a new bot. How are we going to call it? Please choose a name for your bot. 09:44

ARC_EMERGENCY 09:44

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot. 09:44

arc_emergency_bot 09:44

Done! Congratulations on your new bot. You will find it at t.me/arc_emergency_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
```
355554268:AAG1IHv9_sRckMRTAX11nVfbk
RzyG7qjYFY
```

For a description of the Bot API, see this page: https://core.telegram.org/bots/api 09:44

/setjoingroups 09:48

Choose a bot to change group membership settings. 09:48

@arc_emergency_bot 09:48

'Enable' - bot can be added to groups.
'Disable' - block group invitations, the bot can't be added to groups.
Current status is: ENABLED 09:48

Enable 09:48

Success! The new status is: ENABLED. /help

Figure 21: BotFather in action

Once the bot had been created the next step was choosing an API to start developing our bot. We tried a few options but ended up using telebot for reasons mentioned above.

Telebot is a python module and thus it's imported in the usual manner:

```
Import telebot
```

To assign the program to an existing bot we need to use the following instruction:

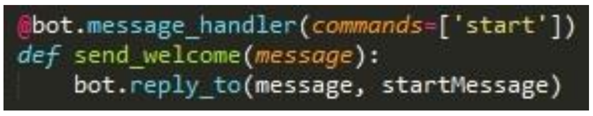
```
bot = telebot.TeleBot(TOKEN)
```

where TOKEN is the bot's API token, given by the BotFather.

And to start the bot:

```
bot.polling()
```

Now, Telegram's Bot API allows the use of decorators to map user's commands with specific methods in the bot's code. This isn't what gives this particular bot its main functionality, but we figured we might as well learn how to use it.



```python
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.reply_to(message, startMessage)
```

Figure 22: using decorators with telebot

In figure 22, the command /start is handled by the method send_welcome, where the bot replies with a starting message.

ARC bot responds to these three commands:

- /start – Get hello message
- /info – Get bot information
- /help – Commands list

What we really wanted our bot to do, however, is to send a custom message to a specific chat. To complete this we needed two things:

- Our bot has to be running.
- An https request has to be made, specifying the bot Token, the chat ID and the message.

The https request is made from both skills that interact with Telegram.

3.2 ARC-Notifications

3.2.1 Introduction

When we started developing the project we quickly ran into a problem. The Echo device lacks push-notifications and thus requires that the user speak first before being able to do anything.

Amazon announced in May 2017 that push-notifications were coming to the Echo sometime this year, however when we first encountered the problem we had no knowledge of this.

We wanted ARC-Medication to inform the user about their pending medication. In other words, we wanted to make a system where Alexa would periodically gather information from the medication calendar and inform the user if they have to take anything at that time.

3.2.2 Objectives

The functional objectives for making ARC-Notifications were the following:

- Automated system.
- Periodically check medication calendar.
- Inform user in case of pending medication.

3.2.3 Development

<u>Technologies used</u>

- Curl
- Pico2wav
- subprocess

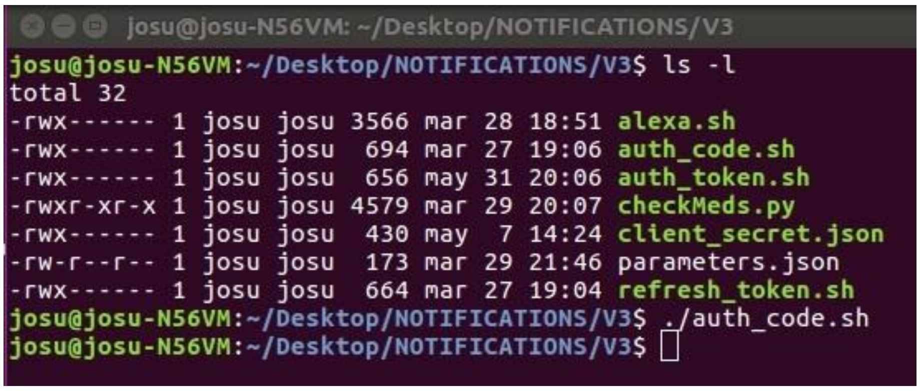
An Echo device couldn't be used for what we had in mind, because we can't program it to say anything without having a person speak to it first.

However, the Alexa Voice Service API allows interaction with the Alexa Voice Service from any device registered on Amazon.

So instead of an Amazon Echo we used a laptop with Ubuntu 16.04.

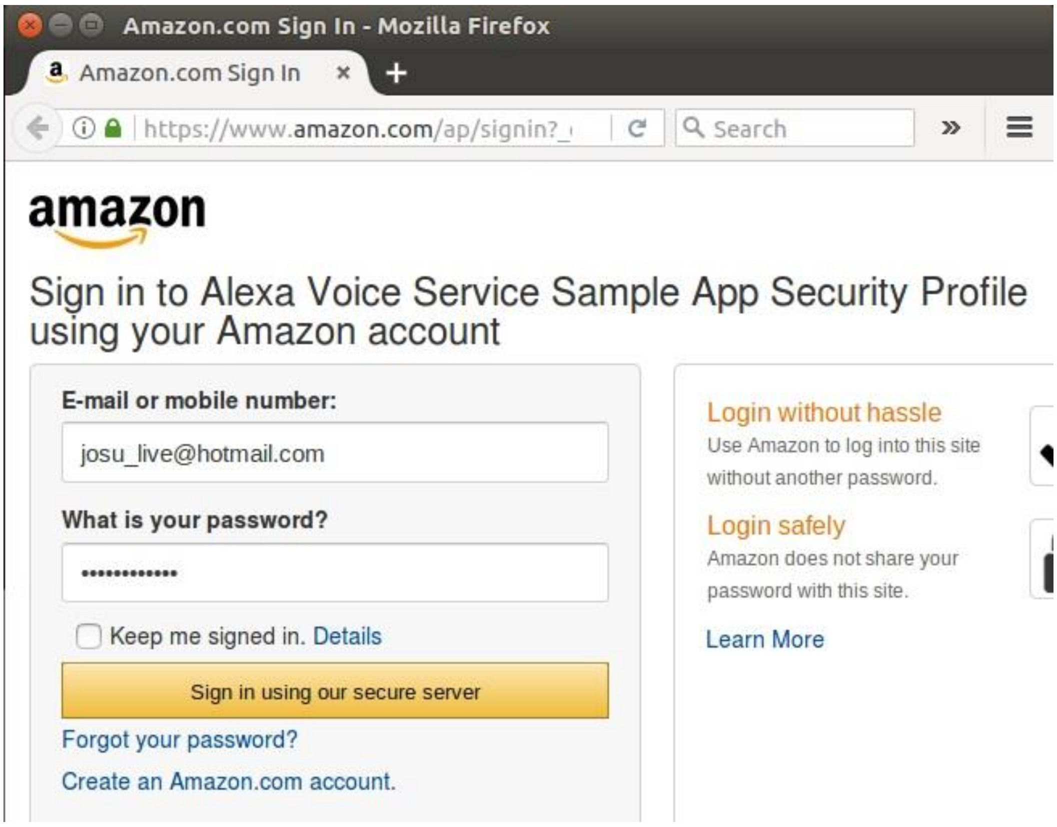
In order to interact with Alexa Voice Service without an Echo device one needs to register a new device in the Alexa Developer Console. Once this is done we will have a Client ID and a Client Secret.

We first need the Client ID to run auth_code.sh (figure 23). This small script generates an authentication URL which should be opened on a browser and then be prompted to login with Amazon (figure 24). The user is then redirected to a redirect URI we have configured after logging in. This redirect URI contains an authentication code as a URL parameter (figure 25).



```
josu@josu-N56VM: ~/Desktop/NOTIFICATIONS/V3
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ ls -l
total 32
-rwx------ 1 josu josu 3566 mar 28 18:51 alexa.sh
-rwx------ 1 josu josu  694 mar 27 19:06 auth_code.sh
-rwx------ 1 josu josu  656 may 31 20:06 auth_token.sh
-rwxr-xr-x 1 josu josu 4579 mar 29 20:07 checkMeds.py
-rwx------ 1 josu josu  430 may  7 14:24 client_secret.json
-rw-r--r-- 1 josu josu  173 mar 29 21:46 parameters.json
-rwx------ 1 josu josu  664 mar 27 19:04 refresh_token.sh
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ ./auth_code.sh
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ 
```

Figure 23: Executing auth_code.sh



Amazon.com Sign In - Mozilla Firefox

Amazon.com Sign In

https://www.amazon.com/ap/signin?_

Search

amazon

Sign in to Alexa Voice Service Sample App Security Profile using your Amazon account

E-mail or mobile number:

josu_live@hotmail.com

What is your password?

•••••••••••••

Keep me signed in. Details

Sign in using our secure server

Forgot your password?

Create an Amazon.com account.

Login without hassle

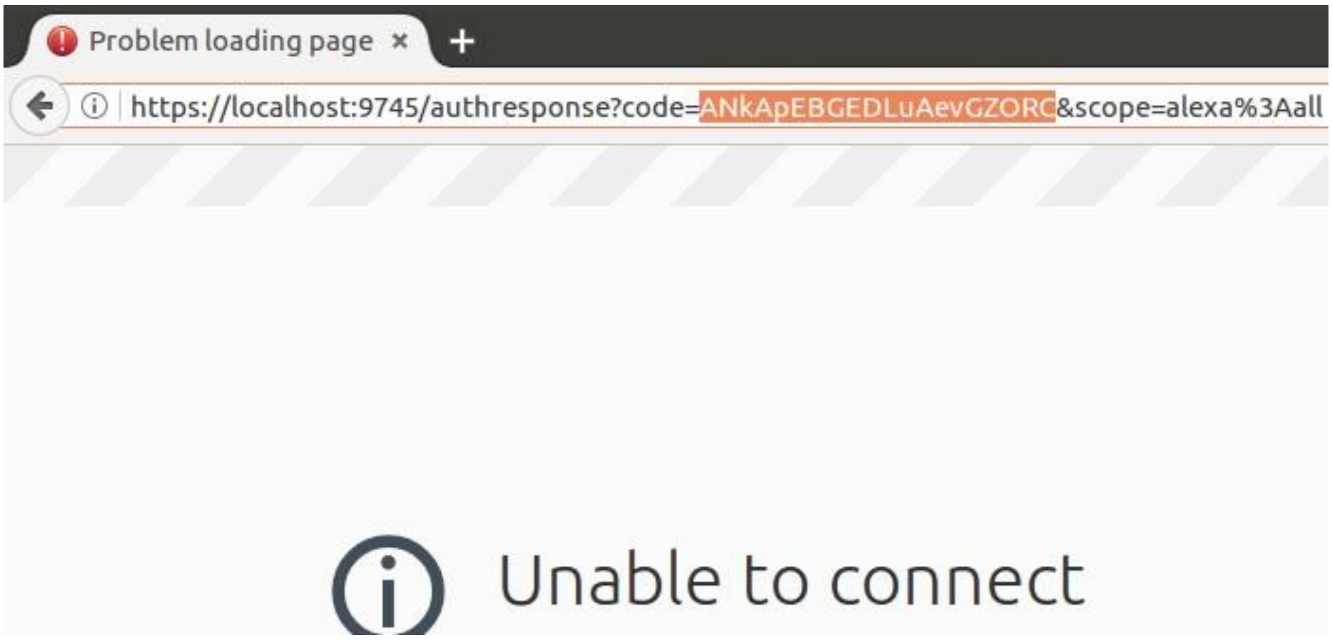
Use Amazon to log into this site without another password.

Login safely

Amazon does not share your password with this site.

Learn More

Figure 24: Amazon Login

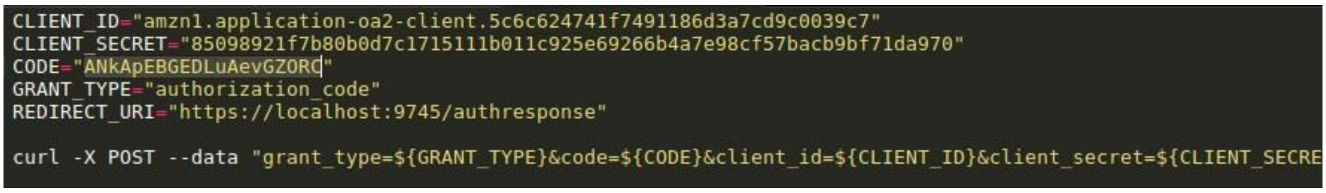

Problem loading page

https://localhost:9745/authresponse?code=ANkApEBGEDLuAevGZORC&scope=alexa%3Aall

Unable to connect

Figure 25: Authentication code

Now, using this code and the client information we got earlier we can run auth_token.sh. This bash script trades the authentication code for an access token in order to make requests to the Alexa Voice Service API from the registered device, in our case a laptop.


```
CLIENT_ID="amzn1.application-oa2-client.5c6c624741f7491186d3a7cd9c0039c7"
CLIENT_SECRET="85098921f7b80b0d7c1715111b011c925e69266b4a7e98cf57bacb9bf71da970"
CODE="ANkApEBGEDLuAevGZORC"
GRANT_TYPE="authorization_code"
REDIRECT_URI="https://localhost:9745/authresponse"

curl -X POST --data "grant_type=${GRANT_TYPE}&code=${CODE}&client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRE
```

Figure 26: auth_token.sh with new authentication code

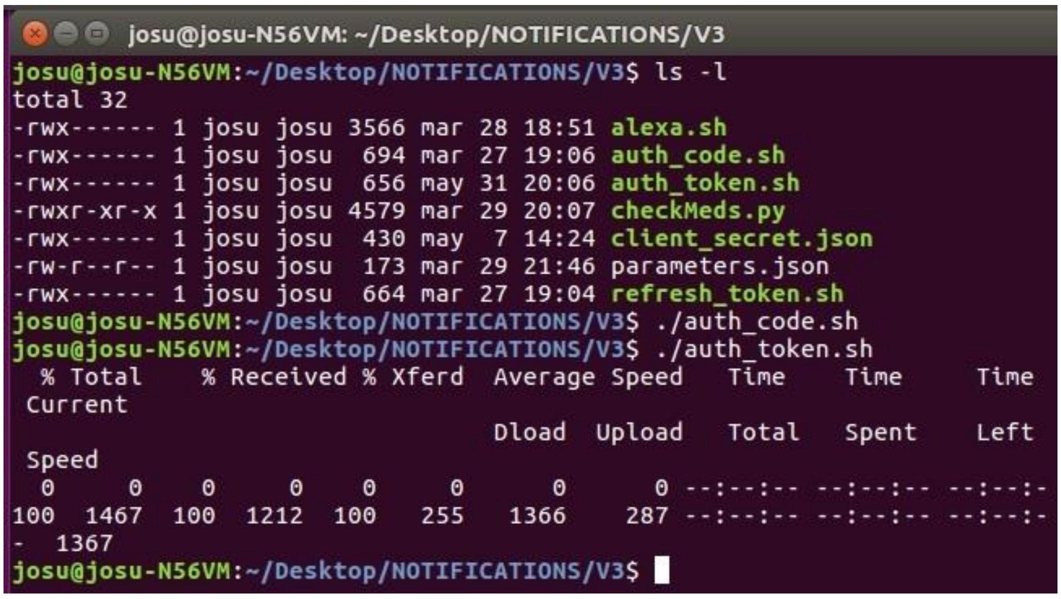
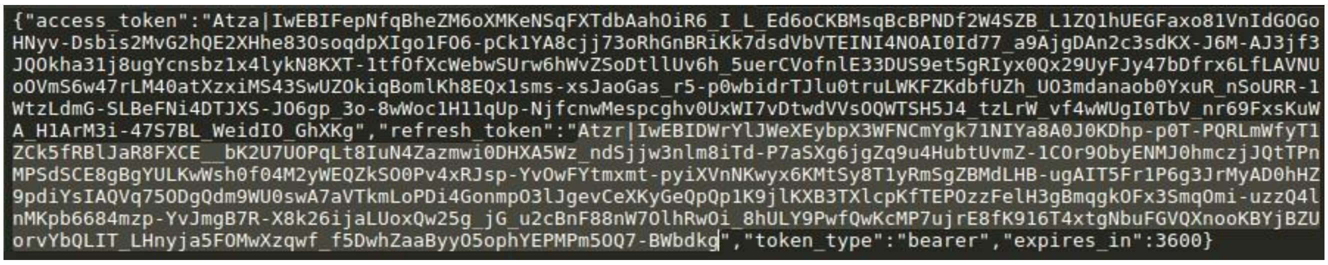

```
josu@josu-N56VM: ~/Desktop/NOTIFICATIONS/V3
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ ls -l
total 32
-rwx------ 1 josu josu 3566 mar 28 18:51 alexa.sh
-rwx------ 1 josu josu  694 mar 27 19:06 auth_code.sh
-rwx------ 1 josu josu  656 may 31 20:06 auth_token.sh
-rwxr-xr-x 1 josu josu 4579 mar 29 20:07 checkMeds.py
-rwx------ 1 josu josu  430 may  7 14:24 client_secret.json
-rw-r--r-- 1 josu josu  173 mar 29 21:46 parameters.json
-rwx------ 1 josu josu  664 mar 27 19:04 refresh_token.sh
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ ./auth_code.sh
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ ./auth_token.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
                                 Dload  Upload   Total   Spent    Left
 Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:-
100  1467  100  1212  100   255   1366    287 --:--:-- --:--:-- --:--:-
-  1367
josu@josu-N56VM:~/Desktop/NOTIFICATIONS/V3$ 
```

Figure 27: Executing auth_token.sh

After running the script we should have an access token and refresh token returned in a JSON response.



```
{"access_token":"Atza|IwEBIFepNfqBheZM6oXMKeNSqFXTdbAahOiR6_I_L_Ed6oCKBMsqBcBPNDf2W4SZB_L1ZQ1hUEGFaxo81VnIdGOGo
HNyv-Dsbis2MvG2hQE2XHhe83OsoqdpXIgo1FO6-pCk1YA8cjj73oRhGnBRiKk7dsdVbVTEINI4NOAI0Id77_a9AjgDAn2c3sdKX-J6M-AJ3jf3
JQOkha31j8ugYcnsbz1x4lykN8KXT-1tfOfXcWebwSUrw6hWvZSoDtllUv6h_5uerCVofnlE33DUS9et5gRIyx0Qx29UyFJy47bDfrx6LfLAVNU
oOVmS6w47rLM40atXzxiMS43SwUZOkiqBomlKh8EQx1sms-xsJaoGas_r5-p0wbidrTJlu0truLWKFZKdbfUZh_UO3mdanaob0YxuR_nSoURR-1
WtzLdmG-SLBeFNi4DTJXS-JO6gp_3o-8wWoc1H11qUp-NjfcnwMespcghv0UxWI7vDtwdVVsOQWTSH5J4_tzLrW_vf4wWUgI0TbV_nr69FxsKuW
A_H1ArM3i-47S7BL_WeidIO_GhXKg","refresh_token":"Atzr|IwEBIDWrYlJWeXEybpX3WFNCmYgk71NIYa8A0J0KDhp-p0T-PQRLmWfyT1
ZCk5fRBlJaR8FXCE__bK2U7UOPqLt8IuN4Zazmwi0DHXA5Wz_ndSjjw3nlm8iTd-P7aSXg6jgZq9u4HubtUvmZ-1COr9ObyENMJ0hmczjJQtTPn
MPSdSCE8gBgYULKwWsh0f04M2yWEQZkSO0Pv4xRJsp-YvOwFYtmxmt-pyiXVnNKwyx6KMtSy8T1yRmSgZBMdLHB-ugAIT5Fr1P6g3JrMyAD0hHZ
9pdiYsIAQVq75ODgQdm9WU0swA7aVTkmLoPDi4GonmpO3lJgevCeXKyGeQpQp1K9jlKXB3TXlcpKfTEPOzzFelH3gBmqgkOFx3SmqOmi-uzzQ4l
nMKpb6684mzp-YvJmgB7R-X8k26ijaLUoxQw25g_jG_u2cBnF88nW7OlhRwOi_8hULY9PwfQwKcMP7ujrE8fK916T4xtgNbuFGVQXnooKBYjBZU
orvYbQLIT_LHnyja5FOMwXzqwf_f5DwhZaaByyO5ophYEPMPm5OQ7-BWbdkg","token_type":"bearer","expires_in":3600}
```

Figure 28: JSON response with access token and refresh token

The refresh token is used to retrieve another fresh access token after it has expired (after 1 hour it’s no longer usable). This is done with refresh_token.sh.

Now that we have an access token we can finally interact with the Alexa Voice Service.

Up until this point we have been following Miguel Mota’s tutorial[32]. He also made a script called alexa.sh that can be used like so:

./alexa.sh “Tell me a joke”

The text given to alexa.sh is converted into a wav file and sent to the Alexa Voice Service. The response is then piped to a ‘play’ command to give the response on the laptop, or the Echo device if it’s used as a Bluetooth speaker.

This is all well and good but we needed something more complex, so we made a python script that uses the Google Calendar API to check for medication events in our medication calendar, only the ones that occur in the current hour. If there is any medication that the patient should take in this hour, it opens a subprocess and runs alexa.sh with the text ‘Simon says you have pending medication’. Simon says is a command that Alexa recognizes and repeats whatever comes next, so Alexa just says ‘You have pending medication’.

Because the process is difficult to setup, a solution is to make a cronjob for refresh_token.sh so that we don’t have to manually get a fresh access token every hour.

If we do that we’re free to interact with the Alexa Voice Service. All we need now is to setup a second cronjob for checkMeds.py to check the medication calendar every hour and notify the user in case of pending medication.

3.2.4 Limitations

This system, while a decent workaround for the lack of push notifications, has its downsides:

- Alexa can give you information, it cannot however ask a question. In specific Flask-ASK terms, it can give a statement but not a question. For us this is an issue because the idea was to notify the patient of their pending medication. When we designed ARC-Medication we wanted Alexa to first ask the whether they had taken today's medication or not. Directly informing the user of their medication was intrusive in our opinion. With this system, the user is notified of the **fact** that they have pending medication, but it is then up to them to launch ARC-Medication to find out what that medication is exactly.
- The system is not executed in a server, it is executed on a PC. Otherwise we can't get audio.
- The computer has to be near the Amazon Echo in order to play the notification audio through it, using it as a Bluetooth speaker. This adds another limitation where the computer running the notification system and the Echo device have to be in the same building, unlike the LAMP server running the rest of the project

3.2.5 Final thoughts on ARC-Notifications

On May 16, not long after developing ARC-Notifications, Amazon announced[33] that 'Notifications for Alexa' feature was coming soon. This didn't really put a wrench in our wheel because:

- As a proof of concept, we were happy with our solution.
- We learned a thing or two along the way.
- Once the tools are out we'll be able to modify our skills to feature push-notifications without the need of this system.

Maybe it wasn't the most productive use of our time but we thought it was an important functionality to feature in the suite, so we're glad we went through with it.

Chapter 4

CONCLUSION

This project aimed at the creation of a set of skills with the Alexa Skills Kit with the objective of providing residential care services for senior patients.

Despite the project currently being in its initial phase, since it's planned to be so much more, we think that the proposed objectives have been achieved. We have developed a firm base that allows incremental additions of functionality as the project grows and expands, as well as providing a starting group of ideas for further inspiration.

Along the way we've encountered issues that hindered our original ideas for ARC but we're satisfied for the time being with the proposed solutions, especially considering that both the Amazon Echo and the Alexa Skills Kit are under constant improvement by the good people at Amazon. Making use of these new functionalities and services will allow ARC to get better over time.

On a personal note I loved developing voice interfaces and figuring out how people interact with machines this way. It helped me understand people a little bit better, and myself as well, so I'm more than glad I took on the project.

Chapter 5

FUTURE IMPROVEMENTS

5.1 Amazon Account Linking

Perhaps the most noticeable improvement that we could add to our project, account linking is the process by which an Alexa user’s account is linked to an external service that also requires authentication.

In our project this external service would be the ARC Control Panel and the database. As it stands all ARC applications retrieve settings from the database, but the only way to know what user ID they need is by hardcoding it into the code. Account linking would solve this problem in an elegant manner.

5.2 Web password hashing

Before ARC is production ready a very important security measure that needs to be taken is to secure the user passwords, using salted password hashing for instance. Securing any ARC setting that might contain sensitive information would also be a good idea.

5.3 Homemade chatbot

For this iteration of ARC-Chatbot we used Amythebot[34]. In an earlier iteration, we used a different chatbot called Mitsuku, but the owner stopped hosting it before we were able to finish our project.

Chatbots come in different flavors, each one specializing on a given functionality.

ARC-Chatbot could benefit from a dedicated chatbot focused on aiding elderly people. This chatbot could be created by modifying an Alicebot using AIML.

5.4 Notifications for Alexa

ARC-Notifications, while a good workaround for a current lack of push-notifications, would not hold up to a proper implementation of the feature. On May 16th Amazon announced that notifications for Alexa was coming soon. This is something that would simplify and improve skills such as ARC-Medication, if implemented properly when the feature is made available.

5.5 Added functionality to ARC-Bot

ARC-Bot could benefit from allowing skill controls, making use of Telegram bot commands. For example, a user could use commands to send a notification to Alexa to ask the user how their day is going. This would allow remote checking of a patient’s state of well-being.

Another option would be to quickly configure skill parameters using the menu interface that Telegram bots feature.

5.6 Added messaging services for ARC-Emergency

Messaging service Whatsapp has an unofficial API called yowsup[35]. It has limited functionality compared to the Telegram API but ARC-Emergency could benefit from a third messaging service option, specially one as popular as Whatsapp.

SMS was also considered in the planning stages of ARC-Emergency. The idea was scrapped due to the cost of sending an SMS message, as the service is not data-based.

5.7 AWS Lambda

Amazon offers a server-less computer service called AWS Lambda that runs your code in response to events and automatically manages the underlying compute resources. It is the recommended hosting option for production-ready skills and the logical next step in developing ARC.

BIBLIOGRAPHY

[1] Flask Python module
http://flask.pocoo.org/

[2] Flask-ASK Python module
https://pypi.python.org/pypi/Flask-Ask

[3] Echosim.io – Alexa Skill Testing Tool
https://echosim.io

[4] LAMP server
http://lamphowto.com/

[5] httplib2 0.10.3
https://pypi.python.org/pypi/httplib2/0.10.3

[6] Python module – os
https://docs.python.org/2/library/os.html

[7] apiclient.discovery 1.0.2
https://github.com/google/google-api-python-client/blob/master/googleapiclient/discovery.py

[8] oauth2client.client 4.1.0
http://oauth2client.readthedocs.io/en/latest/source/oauth2client.client.html

[9] Oauth 2.0
https://oauth.net/2/

[10] Web frameworks
https://www.fullstackpython.com/web-frameworks.html

[11] Flask-ASK Building Responses
https://github.com/johnwheeler/flask-ask/blob/master/docs/responses.rst

[12] Flask-SSLify
https://pypi.python.org/pypi/Flask-SSLify

[13] ssl – TLS/SSL wrapper for socket objects
https://docs.python.org/2/library/ssl.html

[14] MySQLdb
http://mysql-python.sourceforge.net/MySQLdb.html#mysqldb

[15] gettext Python module
https://pypi.python.org/pypi/python-gettext/3.0

[16] datetime
https://docs.python.org/2/library/datetime.html

[17] requests
http://docs.python-requests.org/en/master/

[18] ARC Control Panel url
https://arc.el.ub.edu

[19] StartBootstrap Grayscale Theme
https://startbootstrap.com/template-overviews/grayscale/

[20] Slot Type Reference
https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/built-in-intent-ref/slot-type-reference

[21] NGrok
https://ngrok.com/

[22] Configuring Your Web Service to Use a Self-Signed Certificate
https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/testing-an-alexa-skill

[23] apiclient.errors
https://google.github.io/google-api-python-client/docs/epy/googleapiclient.errors-module.html

[24] MIMEText
https://docs.python.org/2/library/email.mime.html

[25] base64
https://docs.python.org/2/library/base64.html

[26] RFC 3548
https://tools.ietf.org/html/rfc3548.html

[27] dateutil
https://pypi.python.org/pypi/python-dateutil/2.6.0

[28] Beautiful Soup
https://www.crummy.com/software/BeautifulSoup/#Download

[29] Telegram Bot Platform
https://telegram.org/blog/bot-revolution

[30] telebot
https://pypi.python.org/pypi/telebot/0.0.2

[31] The BotFather
https://core.telegram.org/bots#6-botfather

[32] Alexa Notifications using Curl
https://github.com/gravesjohnr/AlexaNotificationCurl

[33] Alexa notifications coming soon
https://developer.amazon.com/blogs/alexa/post/8cc45487-d5fb-413b-b6c7-eeea4794d10c/amazon-announces-notifications-for-alexa-feature-is-coming-soon-sign-up-to-stay-tuned

[34] Amythebot
https://www.pandorabots.com/pandora/talk?botid=878ba74dfe34402c

[35] yowsup (Whatsapp API)
https://github.com/tgalal/yowsup

ANNEXES

Annex A: Skill Conversation Trees

This section features the conversation trees for every ARC skill between Alexa and the user.

It follows this convention:

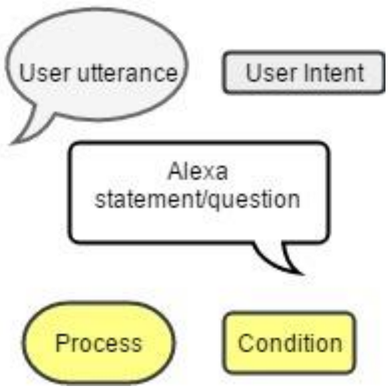


Figure 29: Conversation trees legend

ARC-Emergency

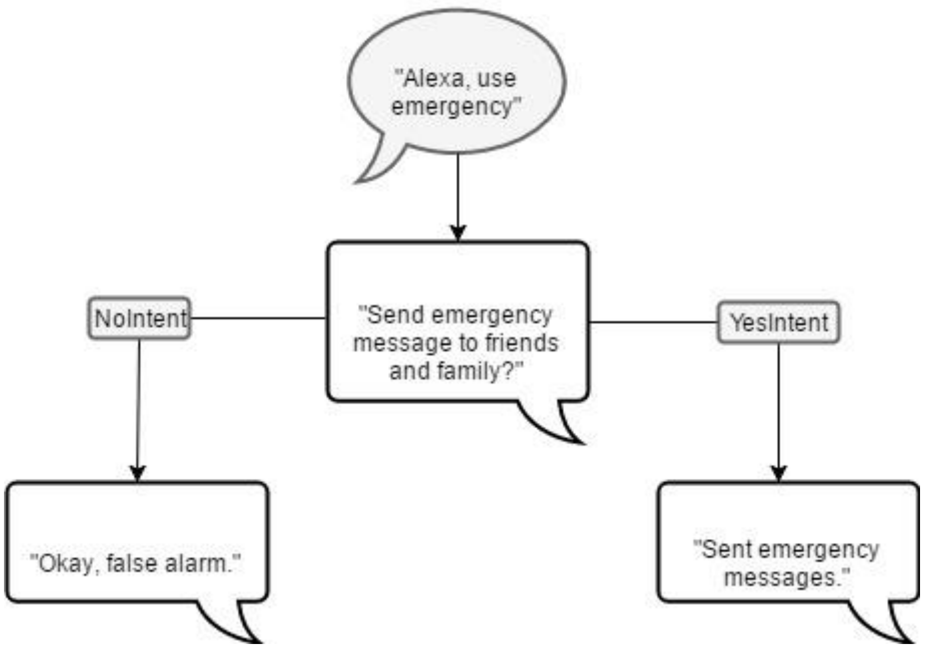

"Alexa, use emergency"
"Send emergency message to friends and family?"
NoIntent
YesIntent
"Okay, false alarm."
"Sent emergency messages."

Figure 30: ARC-Emergency Conversation Tree

ARC-Medication

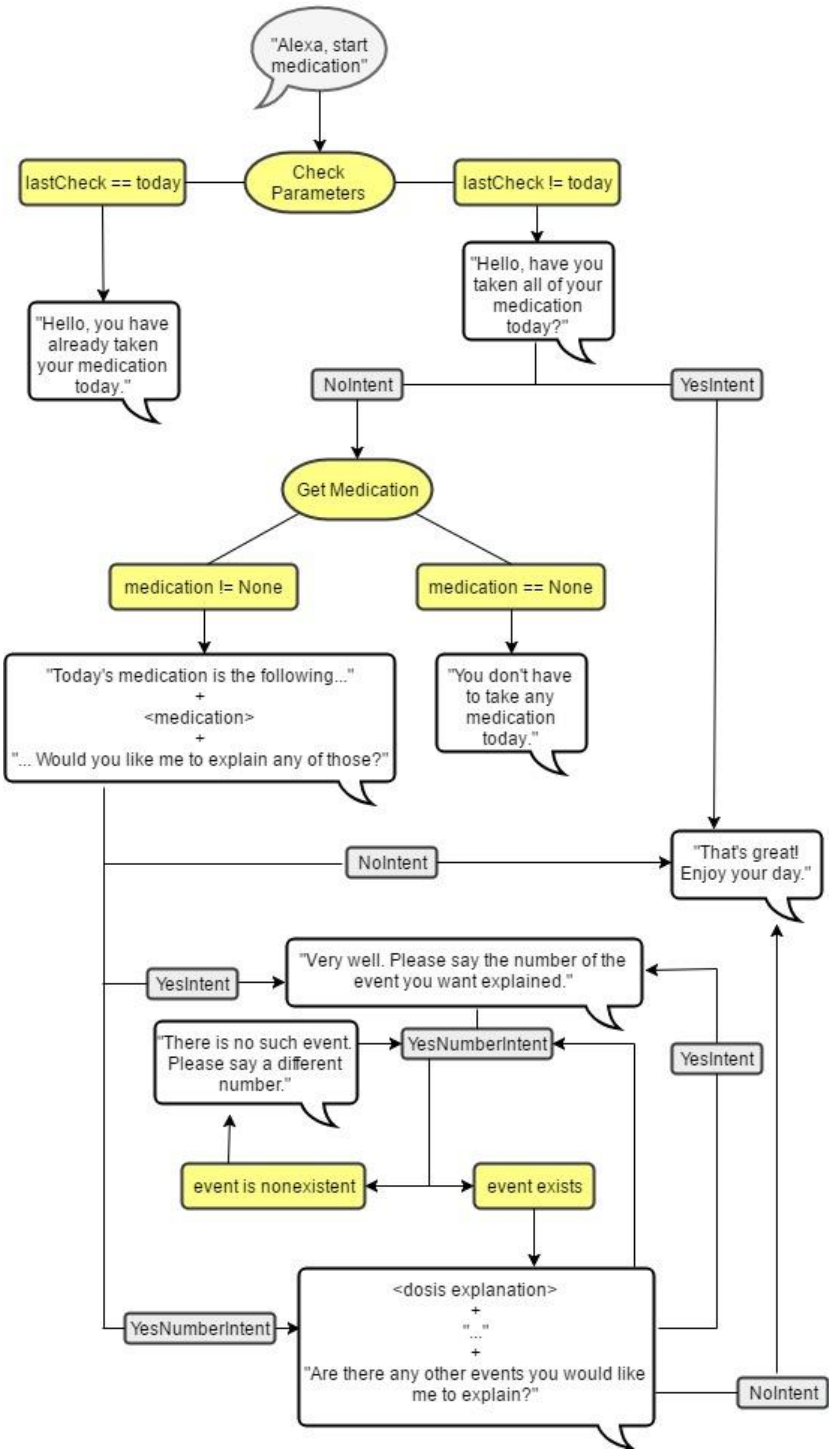

"Alexa, start medication"
Check Parameters
lastCheck == today
lastCheck != today
"Hello, you have already taken your medication today."
"Hello, have you taken all of your medication today?"
NoIntent
YesIntent
Get Medication
medication != None
medication == None
"Today's medication is the following..." + <medication> + "... Would you like me to explain any of those?"
"You don't have to take any medication today."
NoIntent
"That's great! Enjoy your day."
YesIntent
"Very well. Please say the number of the event you want explained."
"There is no such event. Please say a different number."
YesNumberIntent
YesIntent
event is nonexistent
event exists
YesNumberIntent
<dosis explanation> + "..." + "Are there any other events you would like me to explain?"
NoIntent

Figure 31: ARC-Medication Conversation Tree

ARC-Chatbot

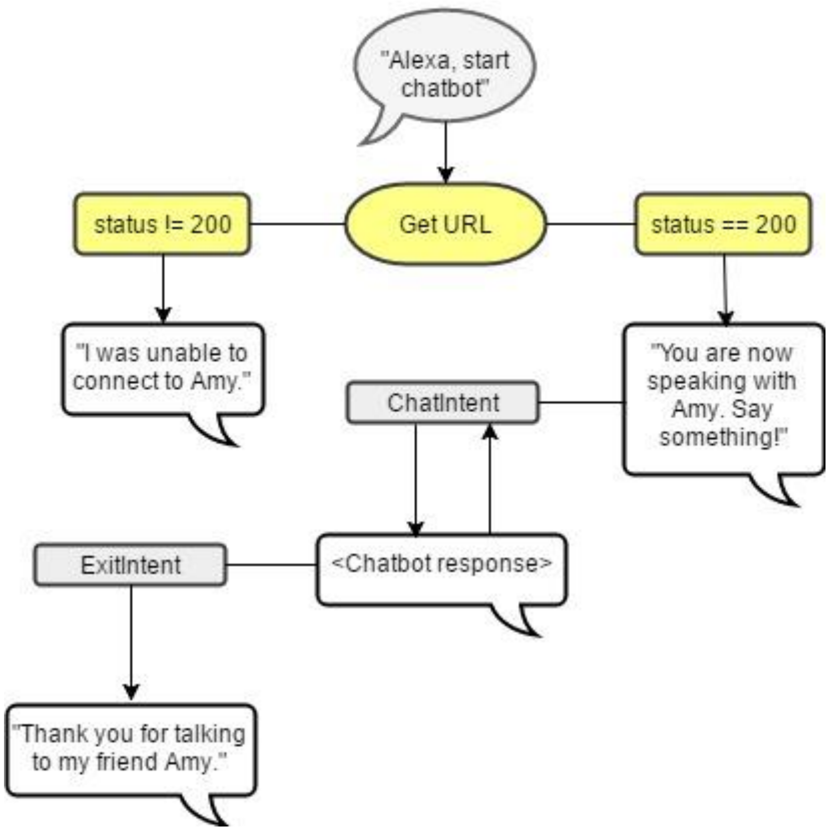

"Alexa, start chatbot"
Get URL
status != 200
status == 200
"I was unable to connect to Amy."
"You are now speaking with Amy. Say something!"
ChatIntent
<Chatbot response>
ExitIntent
"Thank you for talking to my friend Amy."

Figure 32: ARC-Chatbot Conversation Tree

ARC-Telegram

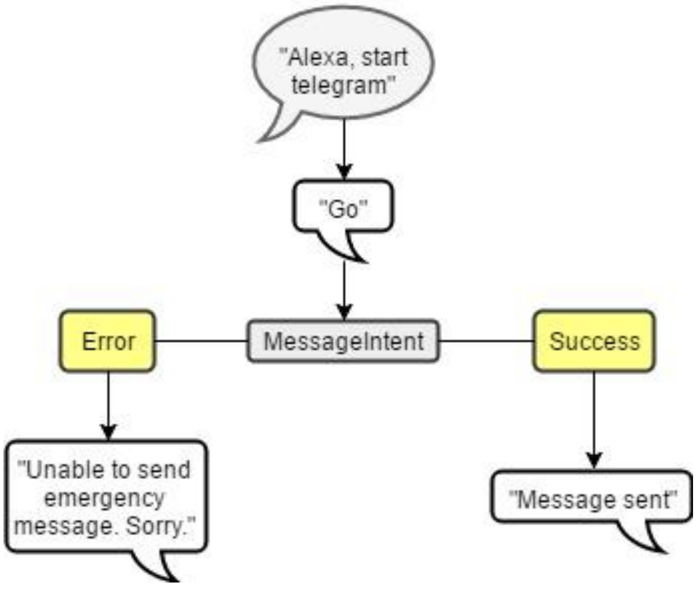


Figure 33: ARC-Telegram Conversation Tree

Annex B: Web Reference

This section showcases the ARC Control Panel views and all available actions in each page.

Login page

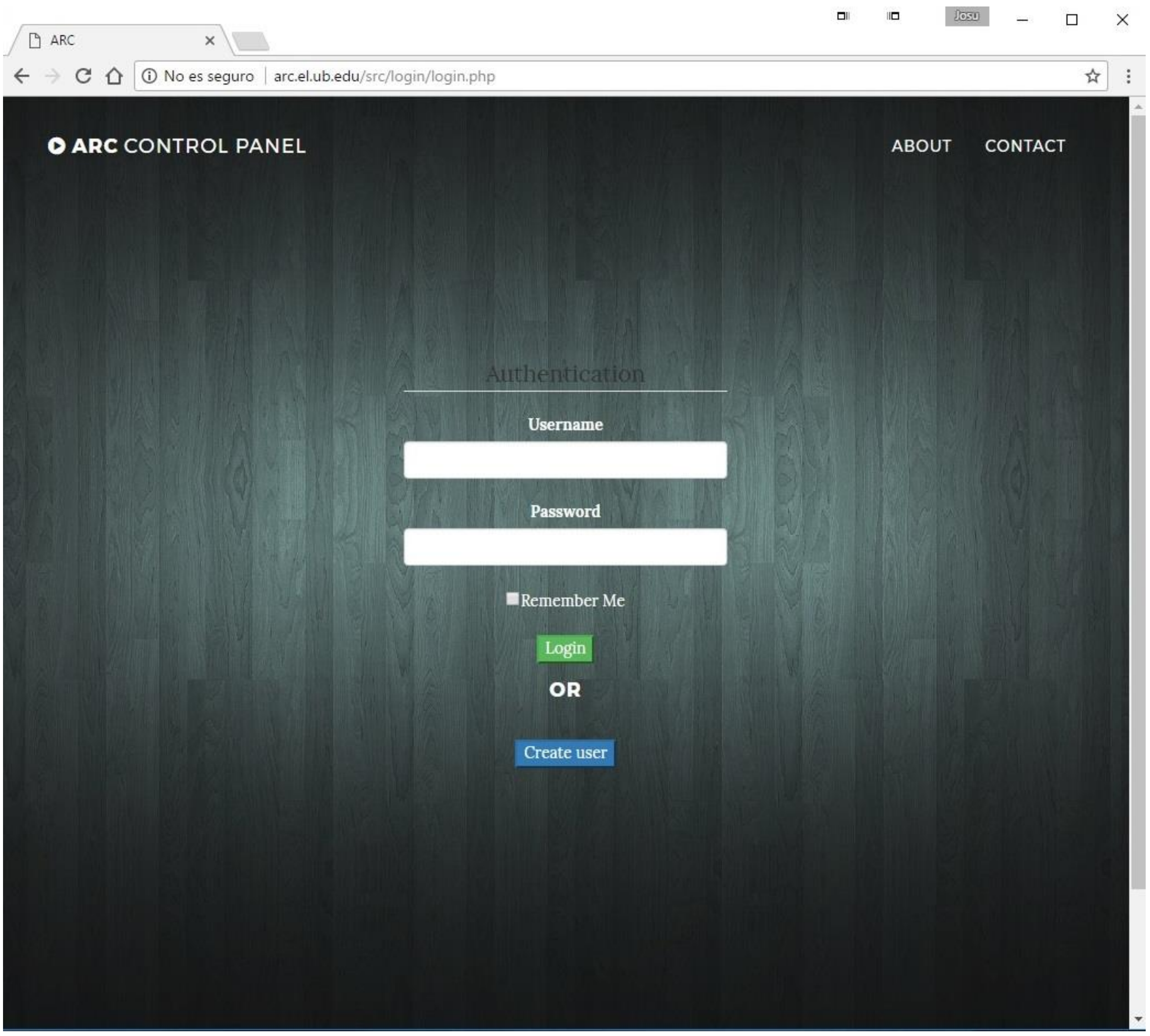


Figure 34: login.php

This page has a form where the user inserts their authentication credentials in order to login into ARC Control Panel. The ‘ABOUT’ and ‘CONTACT’ links are purely aesthetic.

Available actions:

- Login.
- Go to create user page.

Create user page

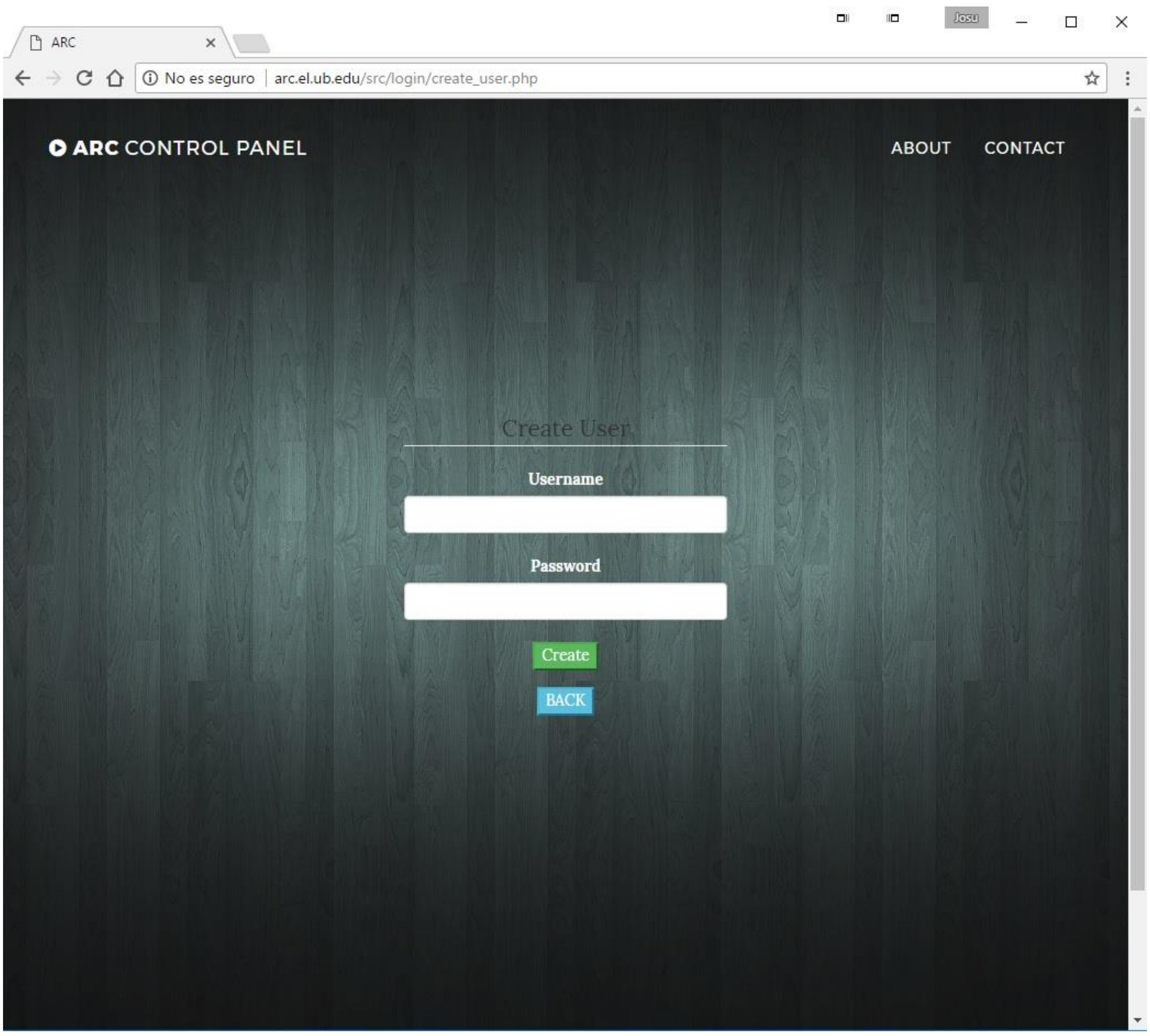


Figure 35: create_user.php

This page contains a form where the user can insert a username and password to create a new ARC user.

Available actions:

- Create user.
- Go to login page.

ARC Control Panel main page

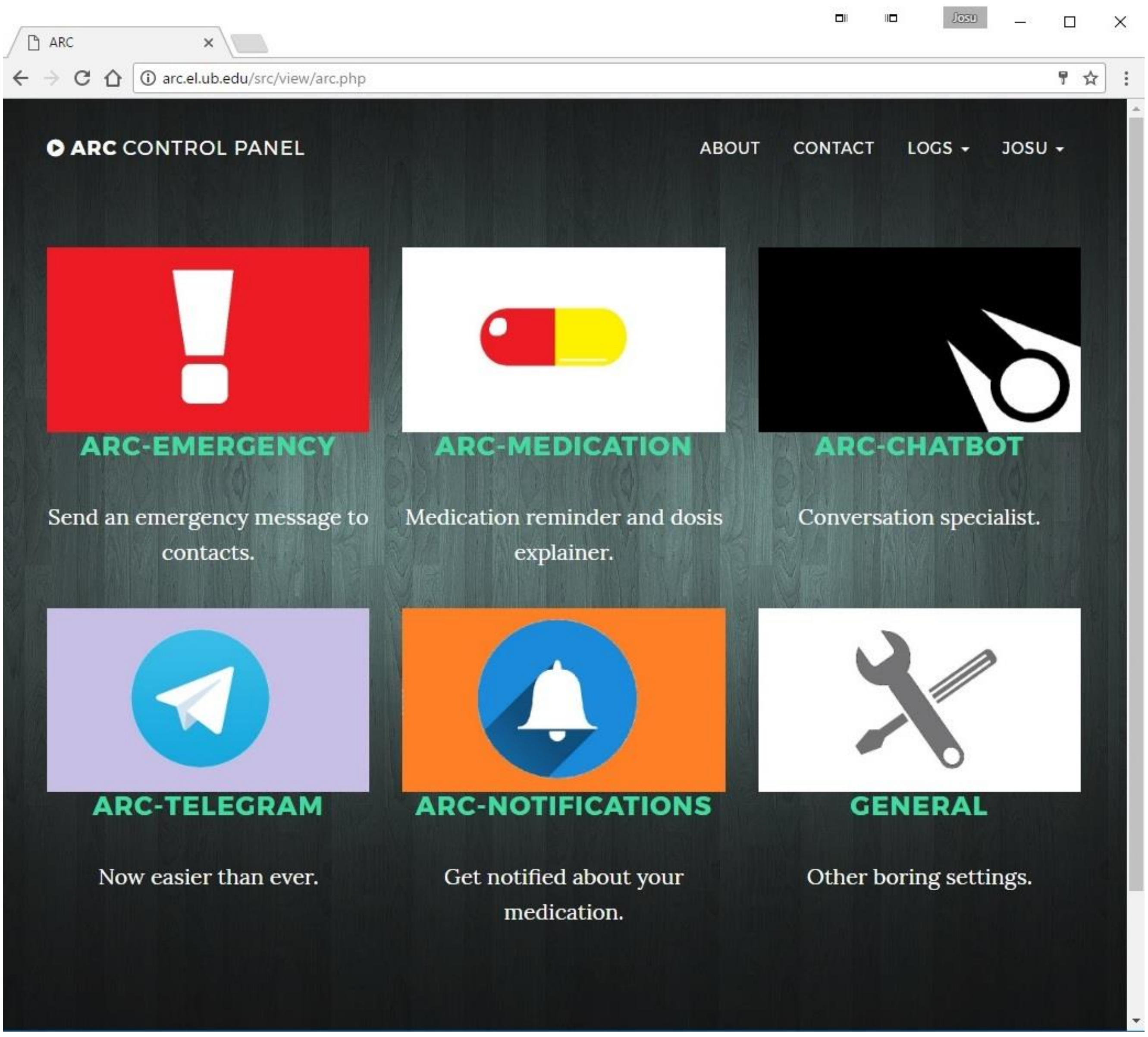


Figure 36: arc.php

From this page the user can choose any skill to edit its particular settings.

Available actions:

- Go to ARC-Emergency configuration page.
- Go to ARC-Medication configuration page.
- Go to ARC-Chatbot configuration page.
- Go to ARC-Telegram configuration page.
- Go to ARC-Notifications configuration page.
- Go to General configuration page.

The user can also use the dropdown menus, available in every page once logged in, to view their logs (shown in Figure 37) and to log out of their session (shown in Figure 38).

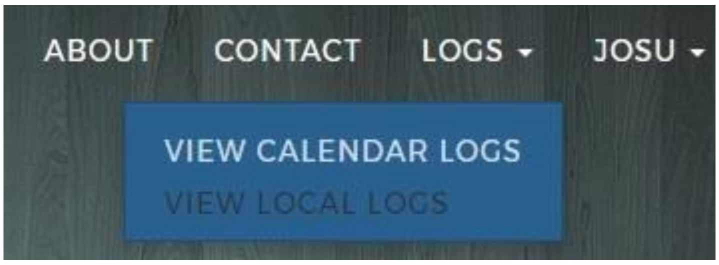


Figure 37:Logs dropdown menu

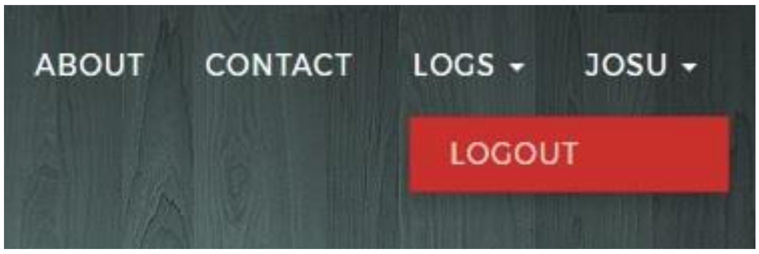
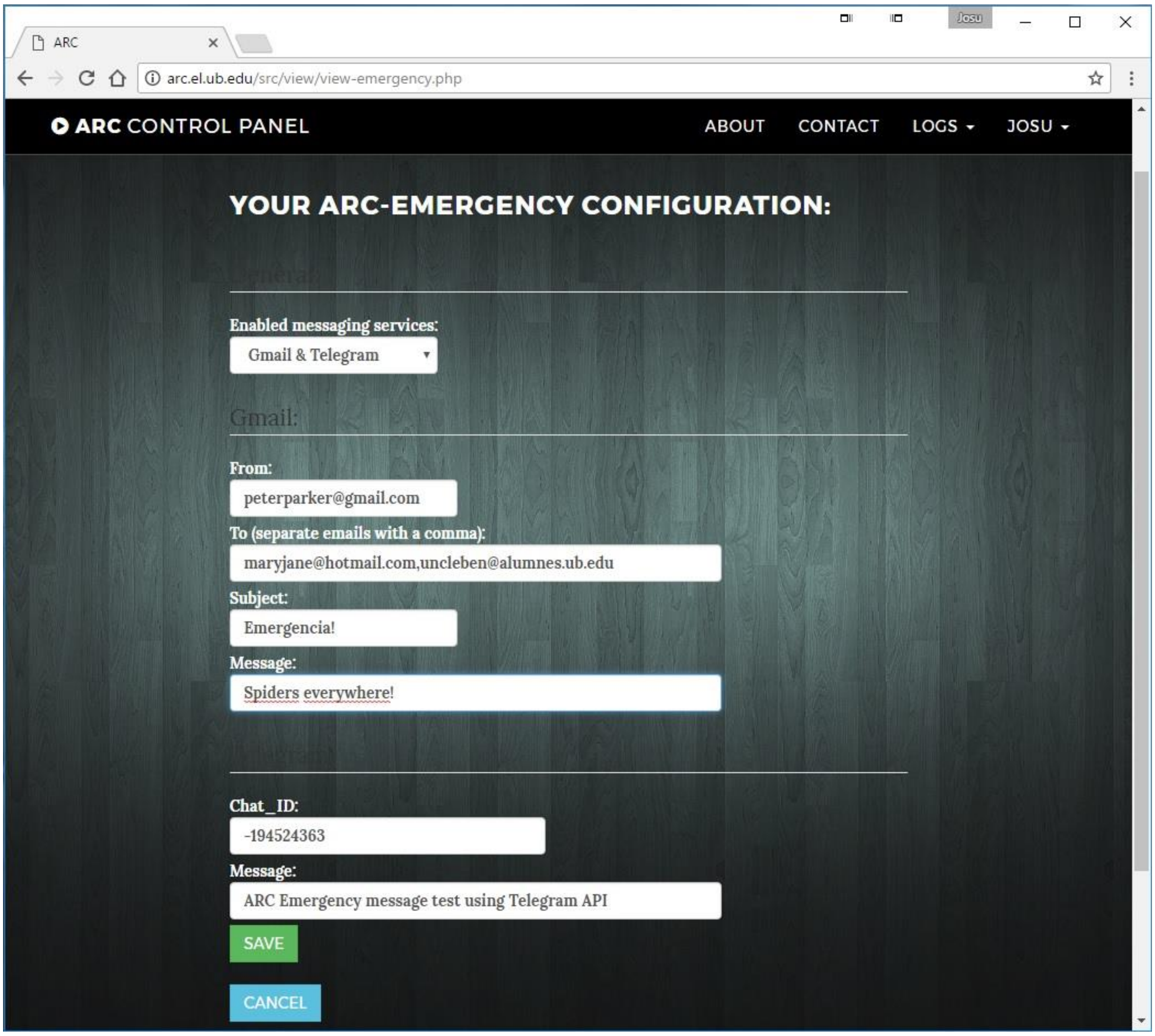


Figure 38:Log out dropdown menu

ARC-Emergency configuration page



YOUR ARC-EMERGENCY CONFIGURATION:

General:

Enabled messaging services: Gmail & Telegram

Gmail:

From: peterparker@gmail.com

To (separate emails with a comma): maryjane@hotmail.com,uncleben@alumnes.ub.edu

Subject: Emergencia!

Message: Spiders everywhere!

Telegram:

Chat_ID: -194524363

Message: ARC Emergency message test using Telegram API

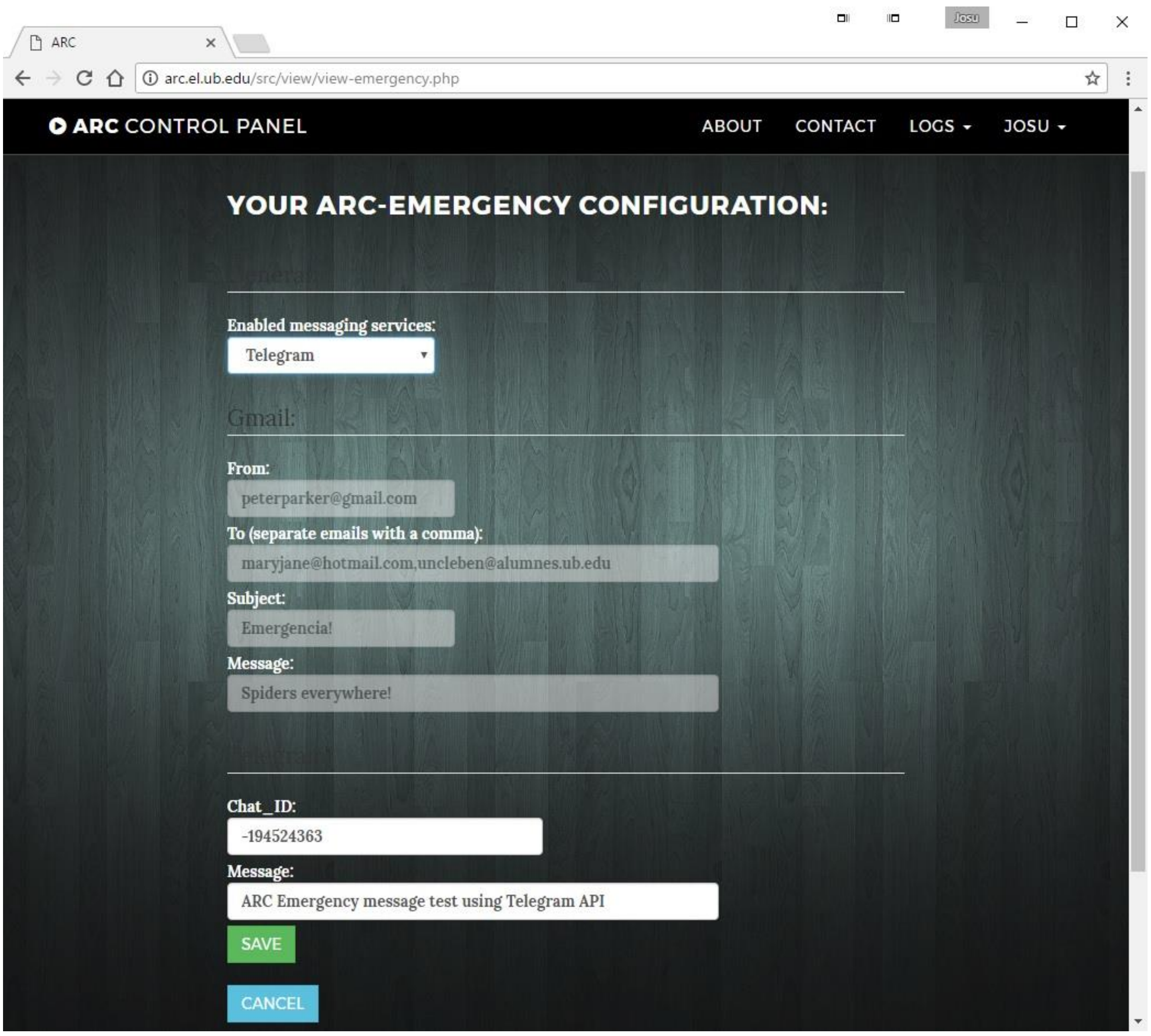
Figure 39:view-emergency.php

This page shows the form containing the user’s ARC-Emergency settings.

Available actions:

- Switch between enabled messaging services.
- Configure skill settings.
- Go to ARC Control Panel main page.

When the user chooses a single messaging service instead of both, the form disables the part of the form corresponding to the other messaging service. This is shown in Figure 40.



YOUR ARC-EMERGENCY CONFIGURATION:

Enabled messaging services: Telegram

Gmail:

From: peterparker@gmail.com

To (separate emails with a comma): maryjane@hotmail.com,uncleben@alumnes.ub.edu

Subject: Emergencia!

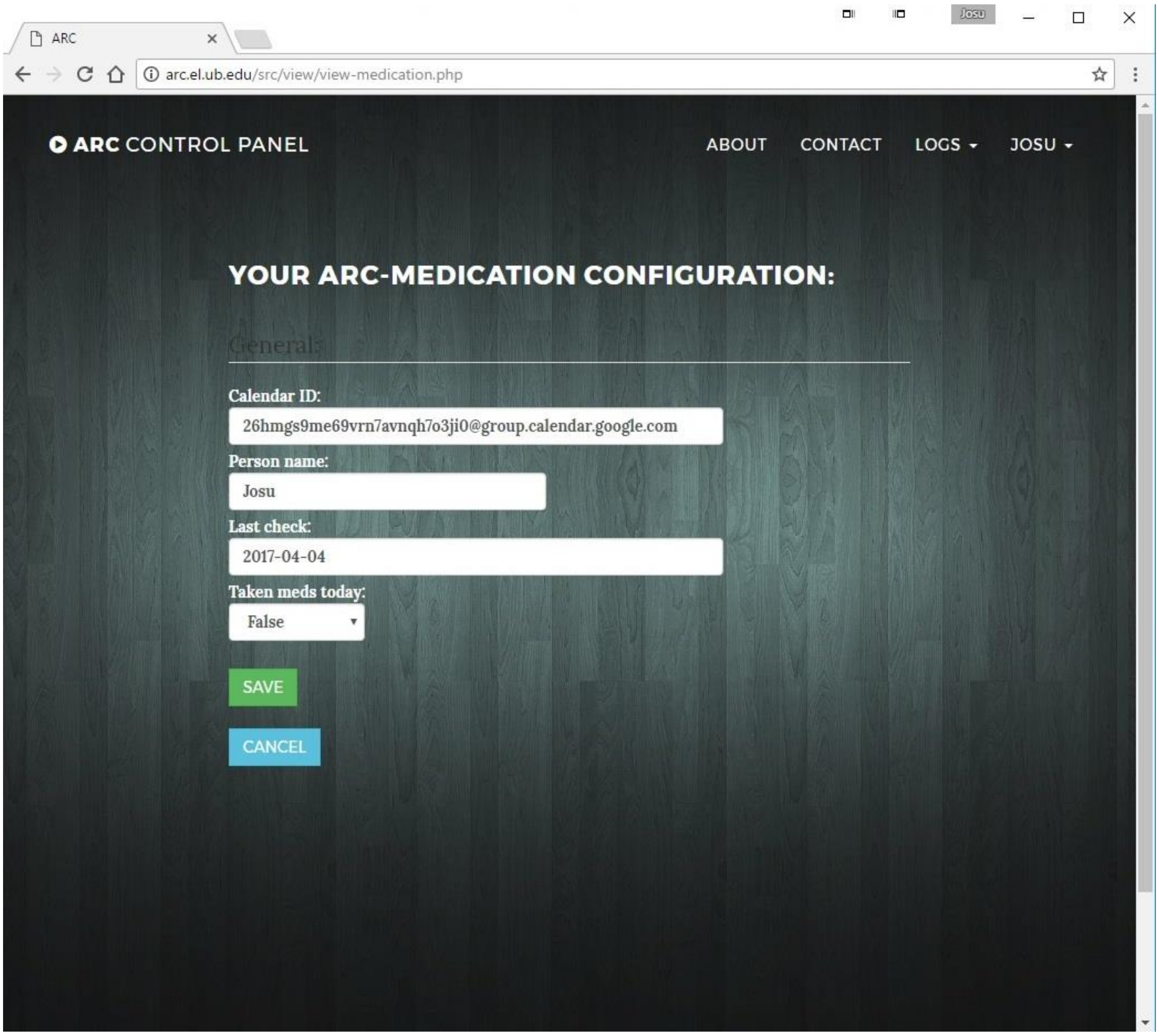
Message: Spiders everywhere!

Chat_ID: -194524363

Message: ARC Emergency message test using Telegram API

Figure 40: Telegram messaging service selected

ARC-Medication configuration page



YOUR ARC-MEDICATION CONFIGURATION:

General:

Calendar ID: 26hmgs9me69vrn7avnqh7o3ji0@group.calendar.google.com

Person name: Josu

Last check: 2017-04-04

Taken meds today: False

SAVE

CANCEL

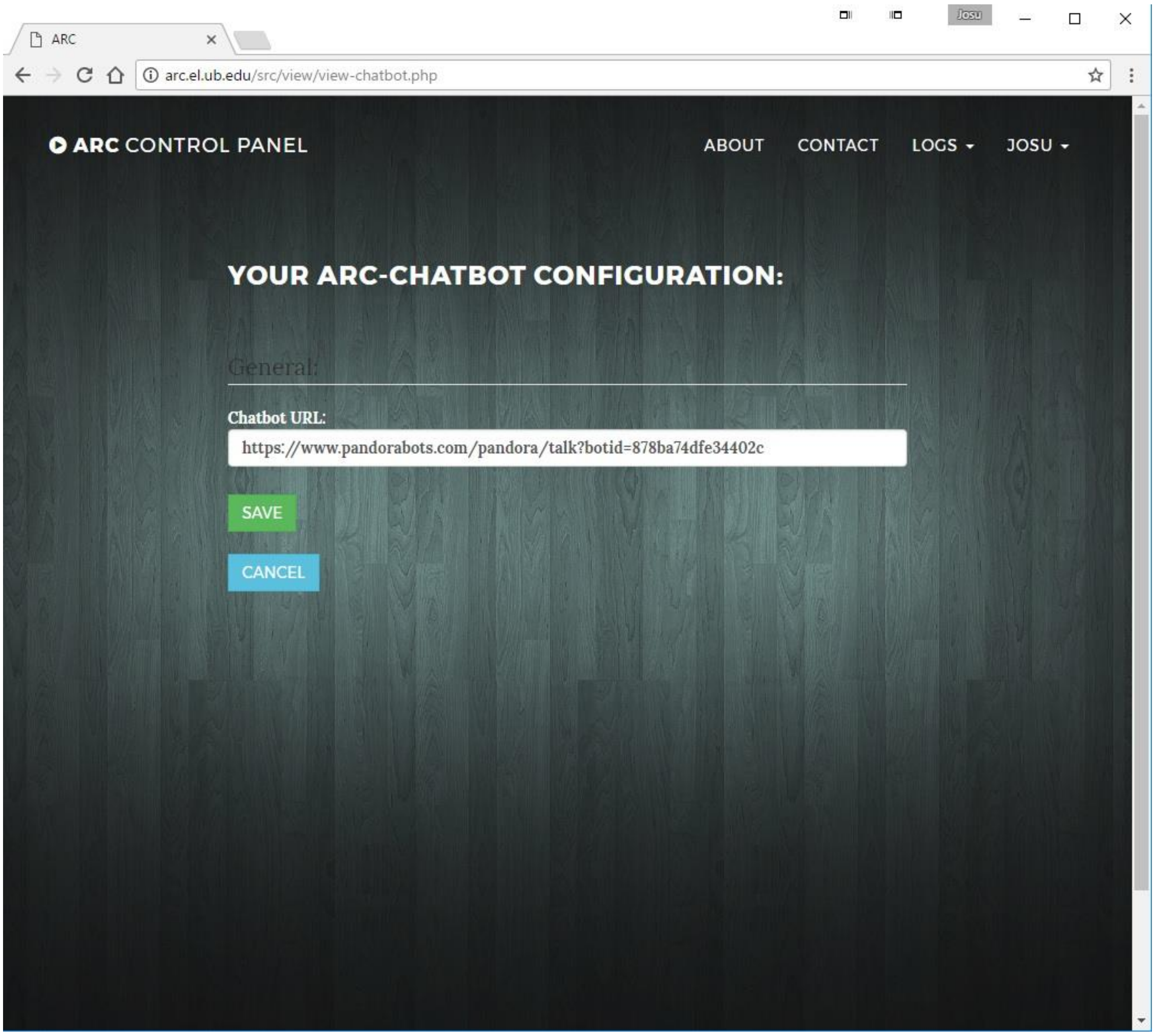
Figure 41: view-medication.php

This page shows the form containing the user’s ARC-Medication settings.

Available actions:

- Configure skill settings.
- Go to ARC Control Panel main page.

ARC-Chatbot configuration page



YOUR ARC-CHATBOT CONFIGURATION:

General:

Chatbot URL: https://www.pandorabots.com/pandora/talk?botid=878ba74dfe34402c

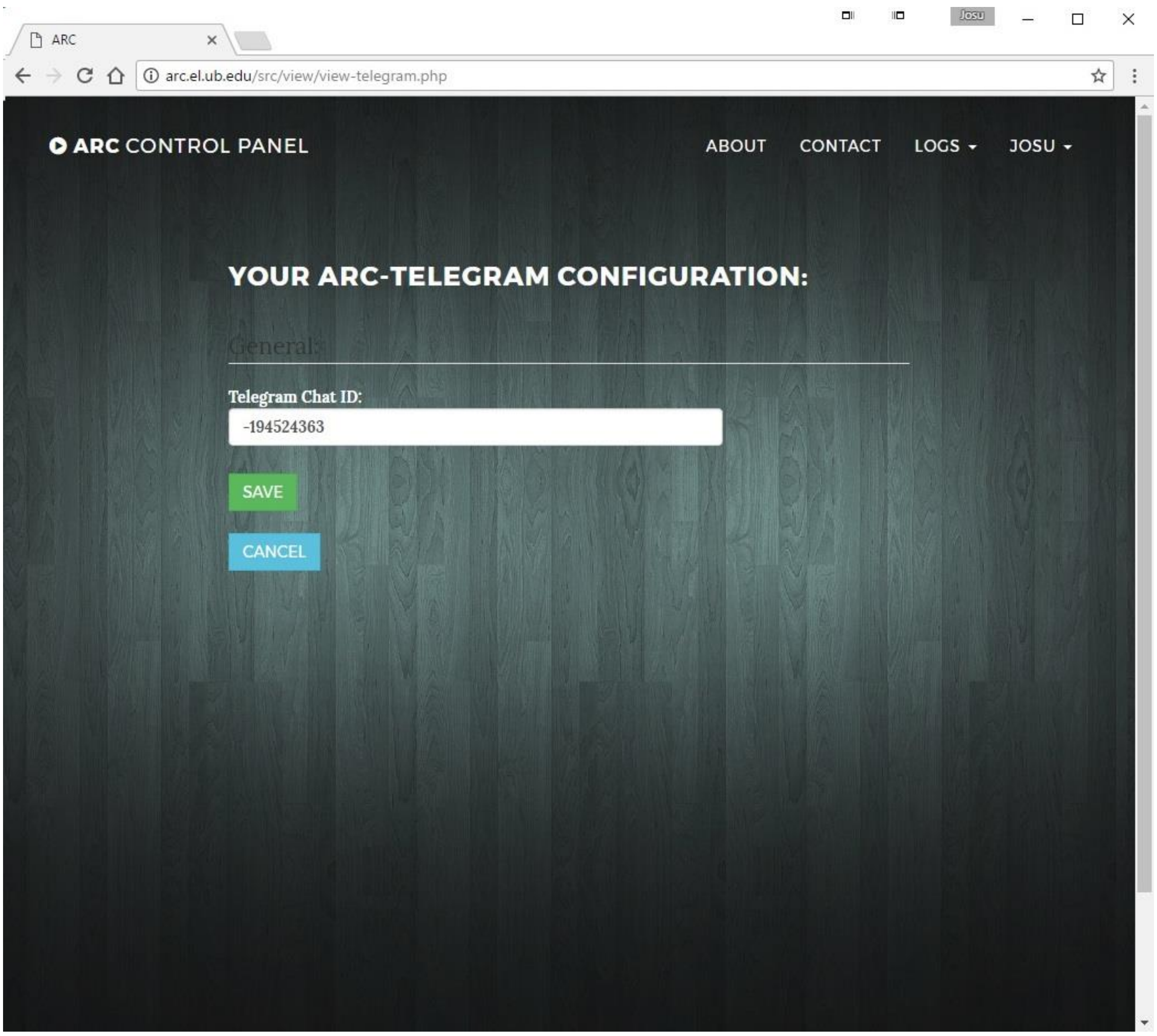
Figure 42: view-chatbot.php

This page shows the form containing the user’s ARC-Chatbot settings.

Available actions:

- Configure skill settings.
- Go to ARC Control Panel main page.

ARC-Telegram configuration page



YOUR ARC-TELEGRAM CONFIGURATION:

General:

Telegram Chat ID: -194524363

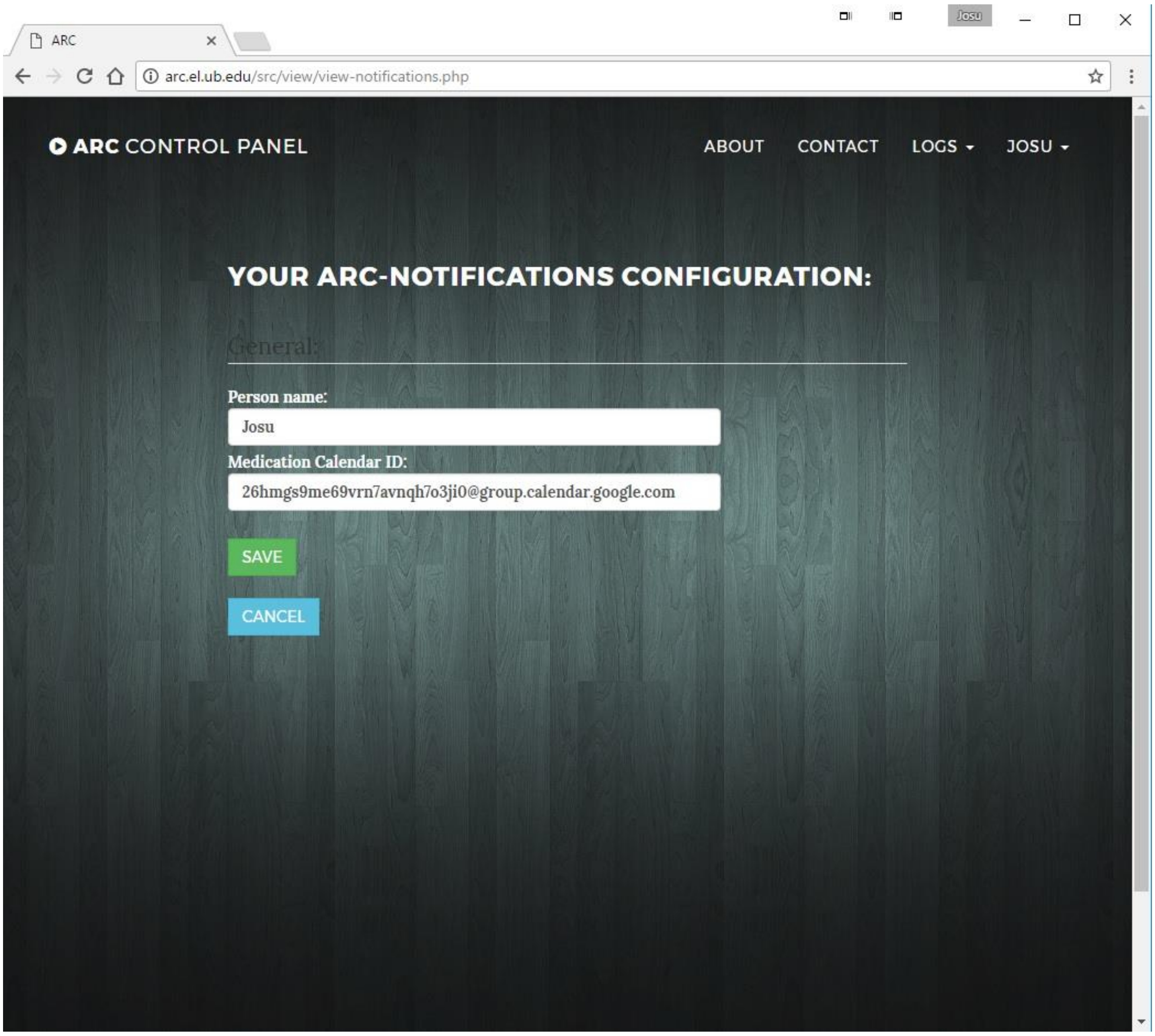
Figure 43: view-telegram.php

This page shows the form containing the user’s ARC-Telegram settings.

Available actions:

- Configure skill settings.
- Go to ARC Control Panel main page.

ARC-Notifications configuration page



YOUR ARC-NOTIFICATIONS CONFIGURATION:

General:

Person name: Josu

Medication Calendar ID: 26hmgs9me69vrn7avnqh7o3ji0@group.calendar.google.com

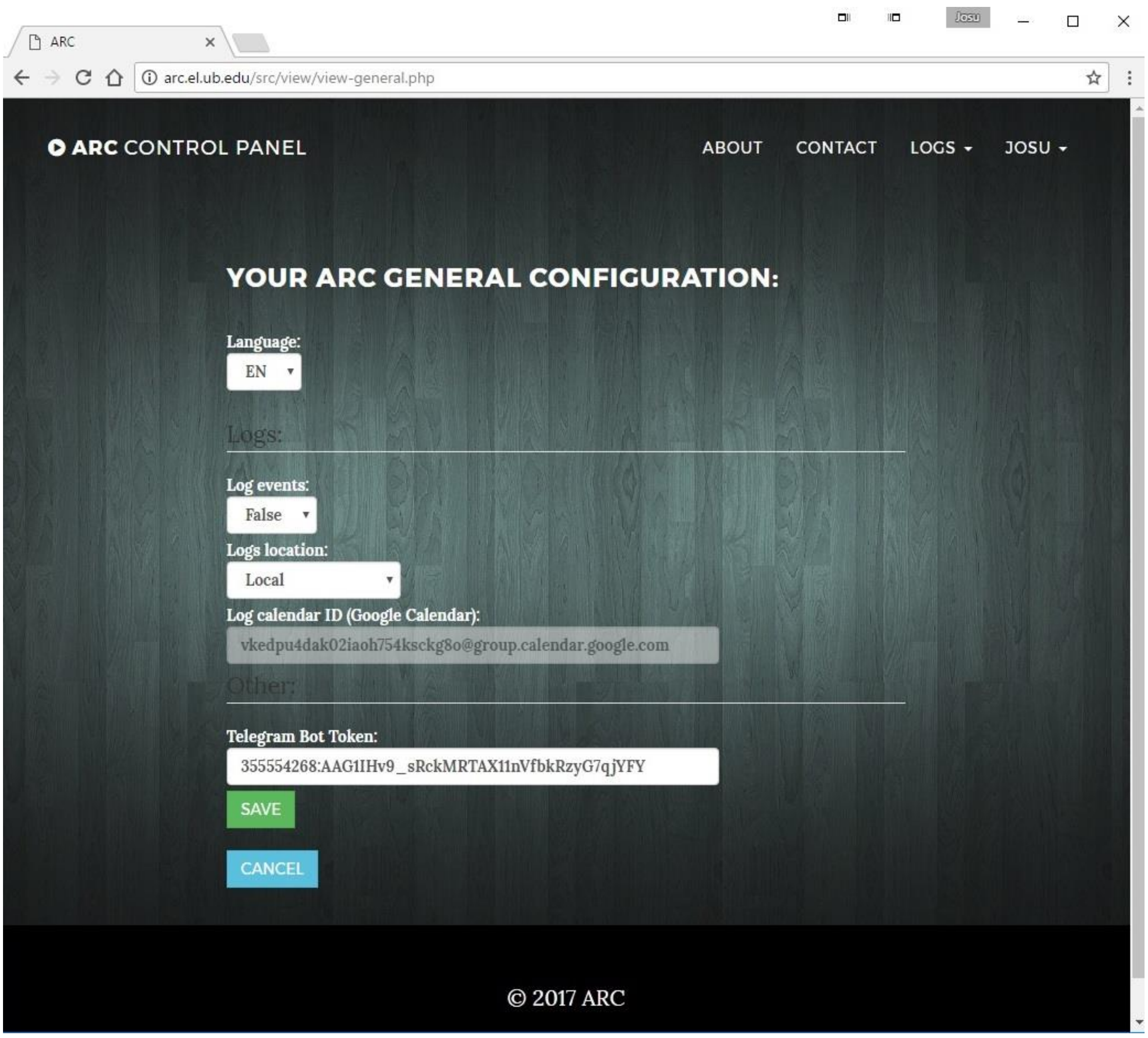
Figure 44: view-notifications.php

This page shows the form containing the user's ARC-Notifications settings.

Available actions:

- Configure skill settings.
- Go to ARC Control Panel main page.

General settings configuration page



YOUR ARC GENERAL CONFIGURATION:

Language: EN

Logs:

Log events: False

Logs location: Local

Log calendar ID (Google Calendar): vkedpu4dak02iaoh754ksckg8o@group.calendar.google.com

Other:

Telegram Bot Token: 355554268:AAG1IHv9_sRckMRTAX11nVfbkRzyG7qjYFY

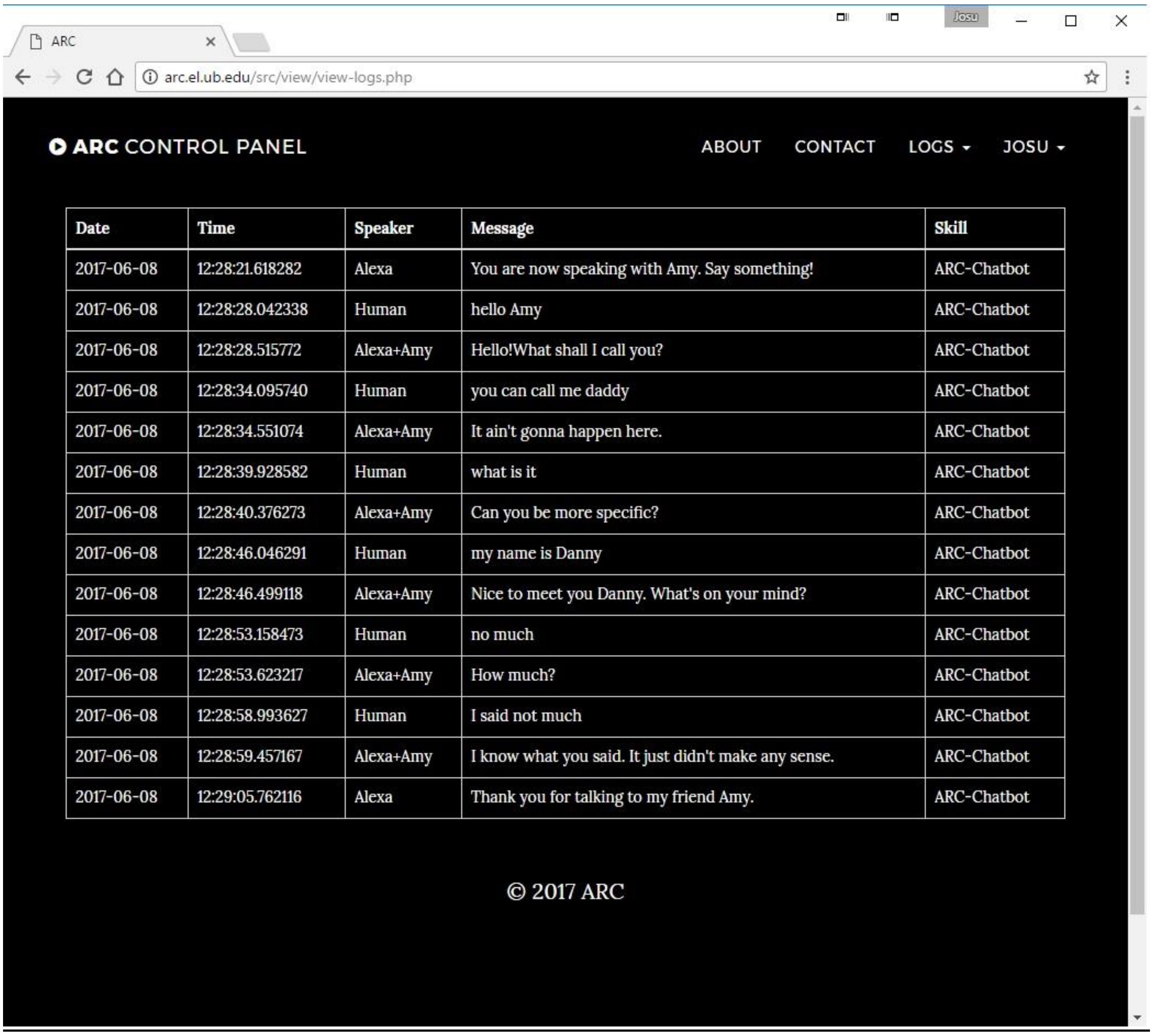
Figure 45: view-general.php

This page shows the form containing the user’s general ARC settings.

Available actions:

- Configure general ARC settings.
- Go to ARC Control Panel main page.

Local logs page



Date	Time	Speaker	Message	Skill
2017-06-08	12:28:21.618282	Alexa	You are now speaking with Amy. Say something!	ARC-Chatbot
2017-06-08	12:28:28.042338	Human	hello Amy	ARC-Chatbot
2017-06-08	12:28:28.515772	Alexa+Amy	Hello!What shall I call you?	ARC-Chatbot
2017-06-08	12:28:34.095740	Human	you can call me daddy	ARC-Chatbot
2017-06-08	12:28:34.551074	Alexa+Amy	It ain't gonna happen here.	ARC-Chatbot
2017-06-08	12:28:39.928582	Human	what is it	ARC-Chatbot
2017-06-08	12:28:40.376273	Alexa+Amy	Can you be more specific?	ARC-Chatbot
2017-06-08	12:28:46.046291	Human	my name is Danny	ARC-Chatbot
2017-06-08	12:28:46.499118	Alexa+Amy	Nice to meet you Danny. What's on your mind?	ARC-Chatbot
2017-06-08	12:28:53.158473	Human	no much	ARC-Chatbot
2017-06-08	12:28:53.623217	Alexa+Amy	How much?	ARC-Chatbot
2017-06-08	12:28:58.993627	Human	I said not much	ARC-Chatbot
2017-06-08	12:28:59.457167	Alexa+Amy	I know what you said. It just didn't make any sense.	ARC-Chatbot
2017-06-08	12:29:05.762116	Alexa	Thank you for talking to my friend Amy.	ARC-Chatbot

© 2017 ARC

Figure 46: view-logs.php

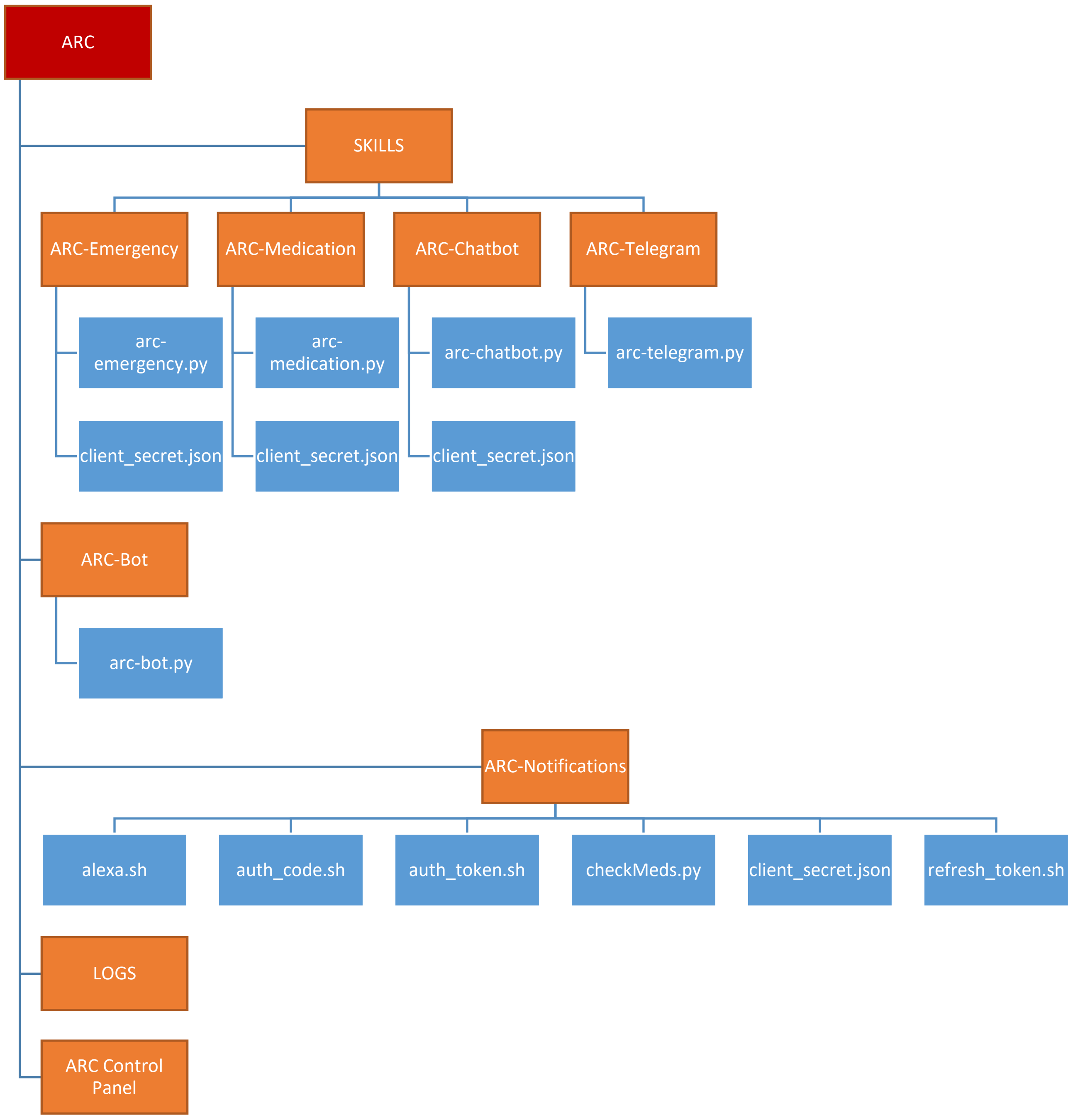
This page shows a table featuring the latest ARC logs stored in the LAMP server.

Available actions:

- Go to ARC Control Panel main page.

Annex C: LAMP Server Project Structure

This is the recommended folder structure for the ARC project within a LAMP server:


ARC
SKILLS
ARC-Emergency
ARC-Medication
ARC-Chatbot
ARC-Telegram
arc-emergency.py
arc-medication.py
arc-chatbot.py
arc-telegram.py
client_secret.json
client_secret.json
client_secret.json
ARC-Bot
arc-bot.py
ARC-Notifications
alexa.sh
auth_code.sh
auth_token.sh
checkMeds.py
client_secret.json
refresh_token.sh
LOGS
ARC Control Panel