

# **A CHATBOT API FOR YIOOP**

**CS 297 Project Report**

**Presented to**

**Dr. Christopher Pollett**

**San Jose State University**

**By**

**Harika Nukala**

**Dec., 2016**

## INTRODUCTION

A chatbot is a computer program that can converse with humans using artificial intelligence in messaging platforms. The goal of the project is to add a chatbot feature and API for Yioop. Yioop is an open source, PHP search engine that can be configured to allow users to create discussion groups, blogs, wikis etc. Yioop provides all the basic features of web search portal. It has its own account management system with the ability to set up groups that have discussions boards. Groups are collections of users that have access to a group feed. The user who creates a group is set as the initial group owner. Posts are grouped by thread in a group containing the most recent activity at the top. The chatbot API for Yioop will allow developers to create new chatbots, powered by rules or artificial intelligence, that can interact like a human with users in a groups feed page. Example chatbots that can be developed with this API is weather chatbots or book flight chatbots.

Over past few years, messaging applications have become more popular than Social networking sites. People are using messaging applications these days such as Facebook Messenger, Skype, Viber, Telegram, Slack etc. This is making other businesses available on messaging platforms leads to proactive interaction with users about their products. To interact on such messaging platforms with many users, the businesses can write a computer program that can converse like a human which is called a chatbot.

Chatbots come in two kinds:

- Limited set of rules
- Machine learning

### **Chatbot that uses limited set of rules**

This kind of bots are very limited to set of texts or commands. They have ability to respond only to those texts or commands. If user asks something different or other than the set of texts or commands which are defined to the bot, it would not respond as desired since it does not understand or it has not trained what user asked. These bots are not very smart when compared to other kind of bots.

### **Chatbot that uses machine learning**

Machine learning chatbots works using artificial intelligence. User need not to be more specific while talking with a bot because it can understand the natural language, not only commands. This kind of bots get continuously better or smarter as it learns from past conversations it had with people.

Here is a simple example which illustrate how they work. The following is a conversation between a human and a chatbot:

**Human:** “I need a flight from San Jose to New York.”

**Bot:** “Sure! When would you like to travel?”

**Human:** “From Dec 20, 2016 to Jan 28, 2017.”

**Bot:** “Great! Looking for flights.”

In order to achieve the ultimate goal, I have taken an iterative approach and divided my work into four major deliverables. These deliverables not only helped me in understanding the code structure of Yioop but also enhances Yioop’s functionality. In the rest of the report, I will be discussing about the four deliverables. To understand more on chatbot service, I had implemented a Facebook Messenger Weather Bot in deliverable 1, which is discussed in next section. The purpose of deliverable 2 is to introduce chatbots to the Yioop. I have added Bot Configuration settings which is used to add bot users in Yioop. In the next deliverable, I have added a functionality where the user will be able to call bots in a group thread. Activation of bots will happen by calling respective callback URL which is already configured that helps bots to have a conversation with users. More details on this is discussed in deliverable 3 section. As a deliverable 4, I have created a weather bot i.e, a web application in php that calls yahoo API to get weather information. The last section of the report contains the conclusion and future work.

## Deliverable 1 - Facebook Messenger Bot

I have implemented a Facebook Messenger Bot to get an overview of how chatbot is build. During this implementation, I understood the flow of control for a chatbot service with other services which is explained below.

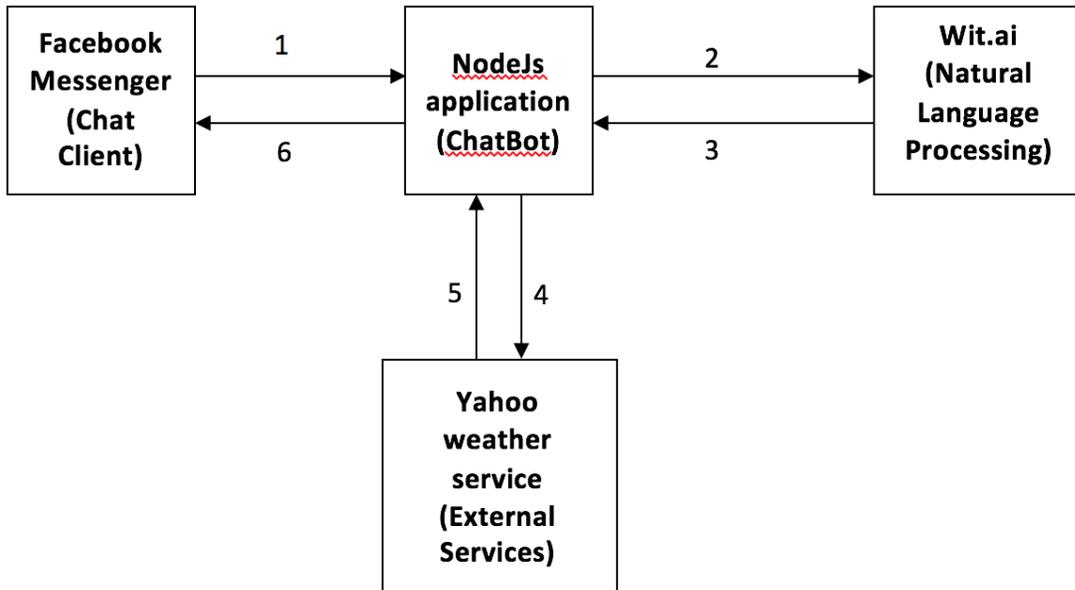


Fig. 1: Architecture of the chatbot

The architecture flow is explained below.

1. User sends message to Chatbot from Facebook Messenger (a Chat Client)
2. Chatbot sends message to Wit.ai (a Machine Learning Natural Language Processing Engine)
3. Wit.ai extracts user's intent and entities from message and sends back to Chatbot
4. Intent is used to call upon external data service to find desired data.
5. The data is returned to chatbot from external service
6. Chatbot builds data into a proper response and sends to Facebook Messenger for display.

In order to create a Facebook Messenger Bot, a developer needs to be authenticated and approved by Facebook to converse with the public and the web server for security reasons. For a Facebook Messenger Bot, I have created a simple web application using Node.js by installing the

necessary dependencies using npm. I ran this locally. I also downloaded and installed ngrok and started it - **npm run ngrok**. This launched a Forwarding URL to the local running server, that means any requests to Forwarding URL will hit the locally running server. This url is used as a Callback URL in Facebook App which will be explained further.

To set up the Facebook App, I have created a Facebook Page and Facebook App using my Facebook account. While setting up a Webhook in the app settings, I have given the Forwarding URL as Callback URL and added code for verification. The access token in page settings is stored as environment variable as it will be used in integration. In order to make webhook to receive messages from this page, the app is subscribed to the page created.

To set up the bot to handle the POST calls at webhook, I have created a webhook endpoint in the sample application.

```
// Message handler
app.post('/webhook', (req, res) => {
  const data = req.body;

  if (data.object === 'page') {
    data.entry.forEach(entry => {
      entry.messaging.forEach(event => {
        if (event.message) {
          const sender = event.sender.id;
          const sessionId = findOrCreateSession(sender);
          const {text, attachments} = event.message;

          if (attachments) {
            //
          } else if (text) {
            //
          }
          else {
            console.log('received event', JSON.stringify(event));
          }
        }
      });
    });
  }
  res.sendStatus(200);
});
```

I also defined constant variables Page\_Access\_Token and Verification\_Token in the application. They are used to verify that the callback is coming from Facebook page. So far I have connected web application to Facebook Messenger using a webhook and have gained access to a facebook page using an access token. Next is to add artificial intelligence to the application using Wit.ai

To set up the Wit.ai, I have created a new app in Wit.ai console which enhances the ability of the bot. The token from app setting is used as WIT\_TOKEN in Node.js application.

The Node-Wit module is used to access Wit application in Node.js application.

```
// bot actions
const actions = {
  send({sessionId}, {text}) {
    const recipientId = sessions[sessionId].fbid;
    if (recipientId) {
      return fbMessage(recipientId, text)
        .then(() => null)
        .catch((err) => {});
    } else {}
  },
  getForecast({context, entities}) {
    return new Promise(function(resolve, reject) {
      var location = firstEntityValue(entities, 'location')
      if (location) {
        //context.forecast = 'sunny in ' + location; we should call a weather API here
        var url = 'https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecas
        request(url, function (error, response, body) {
          if (!error && response.statusCode == 200) {
            var forecast = JSON.parse(body).query.results.channel.item.condition;
            context.forecast=forecast.temp+" and "+forecast.text+" in " +location;
            return resolve(context);
          }
        });
      }
      delete context.missingLocation;
    } else {}
  }
};
```

getForecast is the action method to call Yahoo weather service.

Whenever the user sends a message to the bot in Facebook Messenger, Facebook sends the message to the webhook endpoint, which is nothing but a Node.js application. Then the message is sent to Wit.ai for Natural Language Processing which extracts user’s intent and entities and sends this back to application. This extracted message is then sent to the Yahoo weather API service to find weather information which is returned to the application to build a proper response. This response is then sent to Facebook Messenger for display. The output of this bot is shown in the image below.

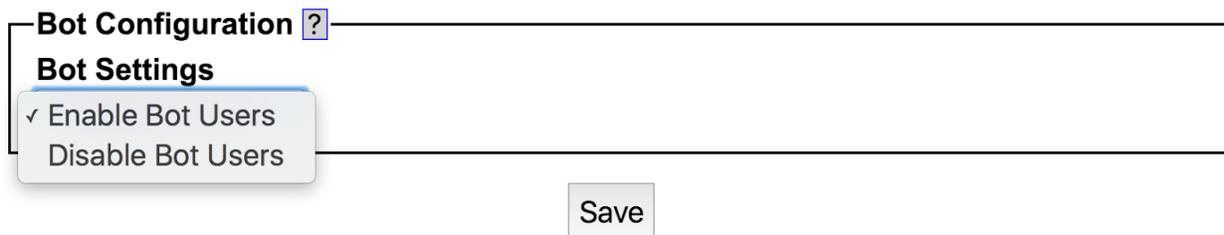


**Fig. 2: Output of Facebook Messenger Weather bot**

## Deliverable 2 – Bot Configuration Settings in Yioop

Yioop can be configured to allow or not to allow users to register for accounts. If allowed, user accounts can create discussion groups. I am implementing the chatbot feature for discussion groups. For such groups, a group owner creates the group and invites other users of Yioop to join. Users can then discuss within the group. I will implement such that a bot user can also join group and responds to the questions asked by normal users. Before implementing this feature, I had to go through Yioop code to understand the code-base and the flow of the Yioop architecture.

We should be able to create accounts for bot users to implement the chatbot feature in Yioop. Currently, Yioop administrator has a section called System Settings where he can enable/disable the specific functionality. I have added an activity where Yioop site administrator should enable bot configuration settings in order to add a bot account. This kind of setting is needed for an administrator where he can enable or disable bot configuration according to his requirement. To enable bot configuration settings, one can select **Enable Bot Users** from the dropdown in the **Bot Configuration** field set of the **Server Settings** activity and save as shown below. This turns on creating bot accounts.



### Create account for bot:

After enabling bot configuration settings, A user can then choose to upgrade the account to bot user in the manage account section. For this functionality, I have added a **Bot User** check box in the **Manage Account** section which appears when a Yioop site administrator has selected the **Enable Bot Users** in the **Server Settings** Activity.

Once the account upgraded as bot user, user should be able to add callback URL and bot unique token as shown in below screenshot. These are the bot settings which will be saved in newly

created **CHAT\_BOT** table by calling *updateBot* in UserModel which inserts new record if it is a new bot else updates existing bot. The callback URL here is similar to Facebook Callback URL which I have mentioned in the Deliverable 1. Whenever a specific bot is called in a group post, a POST request is sent to the callback URL with message and verifies bot with unique token parameter. The image below is the snapshot of a bot user account.

### Account Details

Drag above or ...  
[choose](#) Upload

**Username:**

**First Name:**

**Last Name:**

**Email:**

**Advertise With Us**

**Bot User**

**Bot Unique Token**

**Bot Callback URL**

**Password:**

As we see from the above image, the bot users have many of the same qualities as the normal users: they have profile photos, names, and passwords, they can be mentioned in a post, they can reply to the post messages, and they can be invited to and kicked out of groups. The main difference is the bot users are controlled programmatically by bot unique token and bot callback URL.

## Deliverable 3 – Bot Conversation

### Group Conversation:

A user who joined the group can start a thread with other group members. When a user starts a new thread or comments to an existing thread, several people will be notified via email provided they have their email address configured properly. For a comment to a thread, excluding the commenter, the group's owner, the person who started the thread, and anyone else who has commented on the thread except the bot users would be notified via email. The bot users who have commented on the thread will be notified by hitting callback URL with recent post as a parameter.

In this, I will describe how the bot conversation implementation is done. For testing purpose, I created few bot accounts with unique token and callback URLs. In order to converse with the bot in a group, the bot users must have joined the group. To converse with the bots, user has to mention bot name with '@' in the post (@*bot\_name*). I have implemented a functionality where the bot name is retrieved using regular expression and checks whether requested bot is the bot user (*isUserBot*) and also checks whether it has joined the group.

If it is a bot user and joined the group, then the message after the *bot\_name* in the post is sent to the requested bot callback URL as a parameter. The bot callback URL is called using `FetchUrl::getPages` method which returns the pages from the bot callback URL. These pages are posted as responses by the bot user to the post in a thread. This functionality also includes multiple bot conversation i.e., a user can mention multiple bots in a post so that he can converse with multiple bots in a thread.

```
public function isUserBot($user_id)
{
    $db = $this->db;
    $sql = "SELECT COUNT(*) AS NUM FROM " .
        "USER_ROLE UR WHERE UR.USER_ID = ? " .
        "AND UR.ROLE_ID = ? ";
    $is_bot = false;
    $result = $db->execute($sql, [$user_id, C\BOT_ROLE]);
    if ($result) {
        while ($row = $db->fetchArray($result)) {
            if($row["NUM"] > 0) {
                $is_bot = true;
            } else {
                $is_bot = false;
            }
        }
    }
    return $is_bot;
}
```

```

foreach ($followers as $follower) {
    if($user_model->isUserBot($follower['USER_ID']) == false) {
        $message = tl('social_component_notify_salutation',
            $follower['USER_NAME']) . "\n\n";
        $message .= $body;
        $server->send($subject, C\MAIL_SENDER,
            $follower['EMAIL'], $message);
    } else {
        //call bot URL
        $index = array_search($follower['USER_NAME'],$bots);
        $bots_called[] = $bot_followers[$index];
    }
}
$num_bots = count($bots_called);
$sites = [];
$post_data = [];
for ($i = 0; $i < $num_bots; $i++) {
    $sites[$i][CrawlConstants::URL] = $bots_called[$i]['CALLBACK_URL'];
    $post_data[$i] = "post=$post_followed";
}
$outputs = [];
if (count($sites) > 0) {
    $outputs = FetchUrl::getPages($sites, false, 0, null, self::URL,
        self::PAGE, true, $post_data);
}
foreach ($outputs as $index => $output) {
    $id = $group_model->addGroupItem($parent_item["ID"],
        $group_id, $bots_called[$index]['USER_ID'], $title, $output["q"]);
}

```

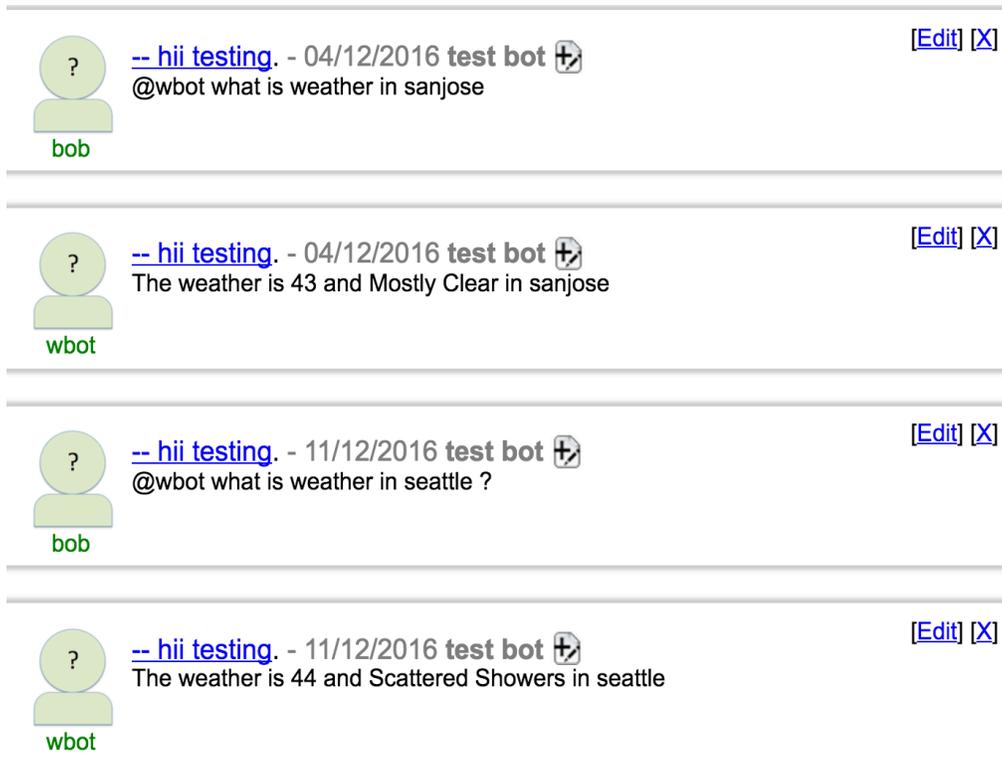
The above is the implementation for the functionality where a user gets notified via email for a new post on a thread if he has already posted something to the thread, and a bot user does not get notified via email but it gets activated by calling its callback URL and new post as its parameter.

## Deliverable 4 – Test weather bot

In this deliverable, I started testing the above implemented functionality by creating a new bot and see how it responds to different inputs. Before implementing bot, you need to figure out what you want your bot user to do. Here I am creating a weather bot to get weather information whenever I ask. For this, I have created a web application that gets weather forecast for a given city from yahoo weather API (external source). It requests yahoo weather API using curl and retrieves the page with the weather forecast details for a given city. The forecast details are returned as JSON object. To display the weather information, the returned JSON object is parsed and retrieved the necessary information which are temperature and condition. The response is constructed properly with temperature and condition of a particular location. On running this application, the weather information is displayed. The code is shown below.

```
/**
 * Get weather information
 *
 * @param string $location the location to get weather updates for
 * @return weather information
 */
function getWeather($location)
{
    $BASE_URL = "http://query.yahooapis.com/v1/public/yql";
    $yql_query = "select * from weather.forecast where woeid in
                (select woeid from geo.places(1) where text='".$location."')";
    $url = $BASE_URL . "?q=" . urlencode($yql_query) . "&format=json";
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $data = curl_exec($ch);
    curl_close($ch);
    $result = json_decode($data);
    $temp = $result->query->results->channel->item->condition->temp;
    $text = $result->query->results->channel->item->condition->text;
    return "The weather is " . $temp . " and " . $text . " in " . $location;
}
```

I have created a bot user and picked username as *wbot* and bot callback URL as the above running application URL. Also the bot user has joined a group. Whenever a user (bob) mention this *wbot* in a group thread, the bot gets activated using bot callback URL which takes message from the user and responds to it very quickly. The image below is the output when testing weather bot.



**Fig. 3: Output of bot conversation in Yioop**

From the above image, we can see that bob (user) asked wbot (bot) about the weather update in San Jose and Seattle and bot responded with the weather information in respective cities.

## **Conclusion:**

Chatbots are the new Apps! As we have discussed in the above deliverables, this project brings the power of chatbots to Yioop and enriches its usability. Chatbots in Yioop can give a human like touch to some aspects and make it an enjoying conversation. And they are focused entirely on providing information and completing tasks for the humans they interact with. The above mentioned functionality in all the deliverables is implemented and pushed in to Yioop code.

By implementing the above mentioned deliverables I was able to add a basic chatbot functionality in to the Yioop. I.e., configuring and creating accounts for bot users with bot settings which is mentioned in deliverable 2, activating a bot whenever a user asks for it via post in a thread which is discussed in deliverable 3 and as I discussed in deliverable 4, I have implemented a simple weather chatbot that gives weather information whenever a user ask and Fig. 3 tells that I was also able to converse with the bot in Yioop. I intend to enhance the system developed so far in CS298. Next step towards building chatbots involve helping people to facilitate their work and interact with computers using natural language or using set of rules. Future Yioop chatbots, backed by machine-learning technology, will be able to remember past conversations and learn from them to answer new ones. The challenge would be conversing with multiple bot users and multiple users.