THE MAKING OF A SUPPORT BOT

HOW DID WE BURST UPON THIS PROJECT?

We, at ERPNext aim at easing the task of managing entrepreneurial resources. Our customers are mainly startups who aim to upgrade their technology of maintaining sales, purchase or employee records at minimal costs. We give them the software, they got to customize it. But how? Go through boring documentations and video lectures, or hire expensive software developers. NO!!! They need an easy live support. May be an assistant, to whom they may ask a question whenever or wherever stuck, and get immediate answers. That could be someone from customer care, or may be a BOT. And, suddenly, a thunderbolt stuck Rushabh's mind. Yeah, why pay the customer care executives when the world is heading towards artificial intelligence. He wanted a bot, a virtual customer care. And then, I joined, as a two month intern, liked this idea and we went on with this.

WHAT EXACTLY WE WANTED TO DEVELOP?

We were crystal clear in this aspect. We wanted a script, which, on getting a query could return a perfect answer. We constructed a database using the posts of the discuss forum of ERPNext and the documentation for frappe. Whenever a user searches for a query, the bot should return a list of best suited posts from this database ranked in order of their suitability. This suitability (or fitness) should depend on various factors like the amount of matches, length of the post, number of likes, reads, incoming links, average length of sentence, average length of word, is it a documentation or a post, is it a reply or a question et al.

Core of the thought was that the top 10 or 15 answers will be selected from the database based upon full text search, and then they will be ranked according to fitness values which are to be predicted by the bot.

WHICH ALGORITHM TO USE?

OK..now thats a pretty tough question. In the initial days, we wandered a lot. Saw numerous online tutorials for AI, learnt the working of existing search forums, and what not. Finally, after a week of study, neural networks occurred. A blessing in disguise, it was an algorithm with the exact same functioning we wanted our bot to aspire. And hence, we narrowed down, neural networks it was.

WHY IS NEURAL NETWORK SUITABLE FOR US?

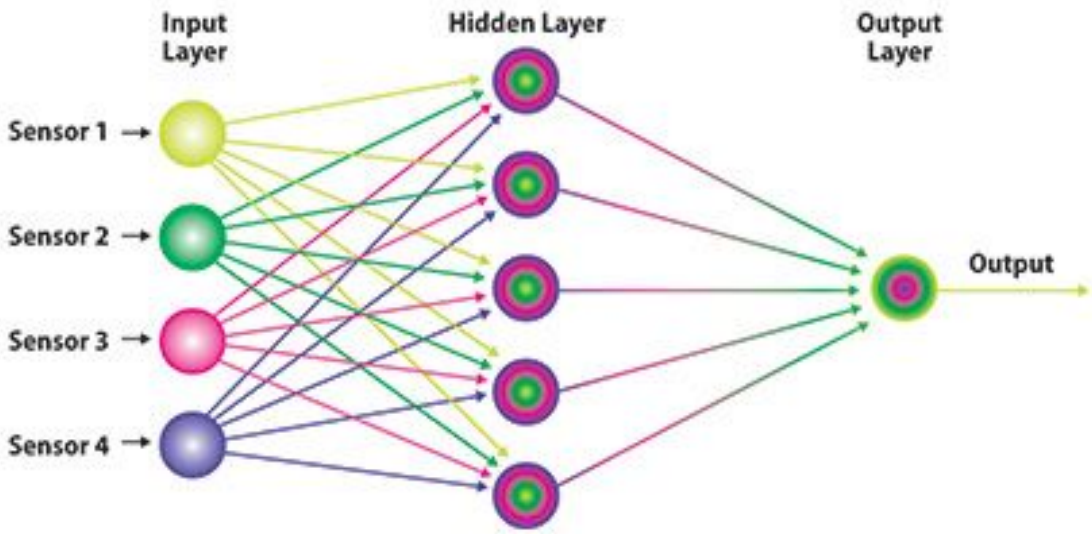
We first came across the idea of neural networks here. We could easily get the clear idea of the main concept behind neural networks. It works in exactly the same way we want our bot to work. It takes the input in terms of certain parameters, weighs each of them, even calculates the nonlinear relationships between them, and hence combines them all in different proportions to give the final output. This is similar to our problem, where we wanted to give a fitness value to each our posts depending upon different parameters.

WHERE DID WE LEARN NEURAL NETWORKS FROM?

We got the initial idea of what neural networks are from here.This is way too abstract, and just by seeing or playing with it, we could not guess the real code (yeah, tensorflow is an open source library, but we at erpnext believe in working from scratch). And then this came under the eyes. It is beautiful. A way too elegant implementation of neural networks in a very simple Python program with ample explanations. Yes, we went ahead with it, but seems this was not the correct sink, the error upon our database went to a soaring value of 60%. Nope, thats not our thing. We were confused, as why is this wrong, and what else did we need to do. And then, this and this happened. We went ahead, and yeah, it worked! Pretty well, now we are at an approximate error of 11%. It took us 6 weeks and a lot of reading and googling to figure out what exactly is the neural network and how do we design it. And thus, here I am, to help out all those numbing across google for this particular implementation of neural networks.

WHAT IS A NEURAL NETWORK?

Well, what wikipedia says is, “In machine learning and cognitive science, **artificial neural networks** (**ANNs**) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) which are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown.”
Yeah, that was a bit complicated. Going easy, a neural network is a simple collection of nodes and edges, where nodes are grouped into layers, and edges are drawn to connect different layers. These nodes are called neurons



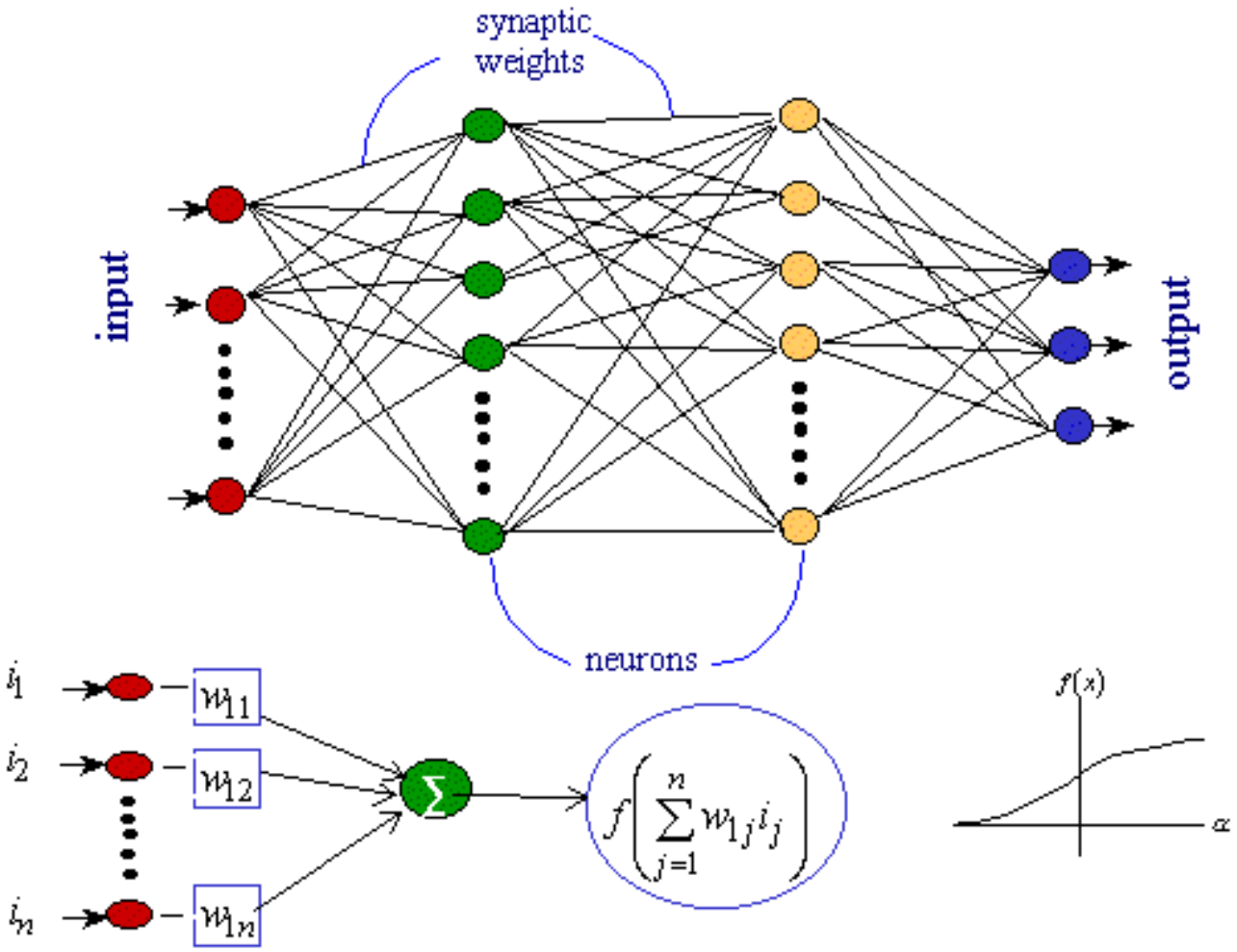
This is a neural network with 3 layers viz Input Layer (4 neurons), Hidden Layer (5 neurons) and the Output Layer with (1 neuron). Let us now see how characteristics are transferred from input to output layer via the hidden layers.

In a neural network, each edge has a weight which signifies the amount of the characteristics of the source node which are to be taken into account at the destination node. These weights on edges are called synapse values.

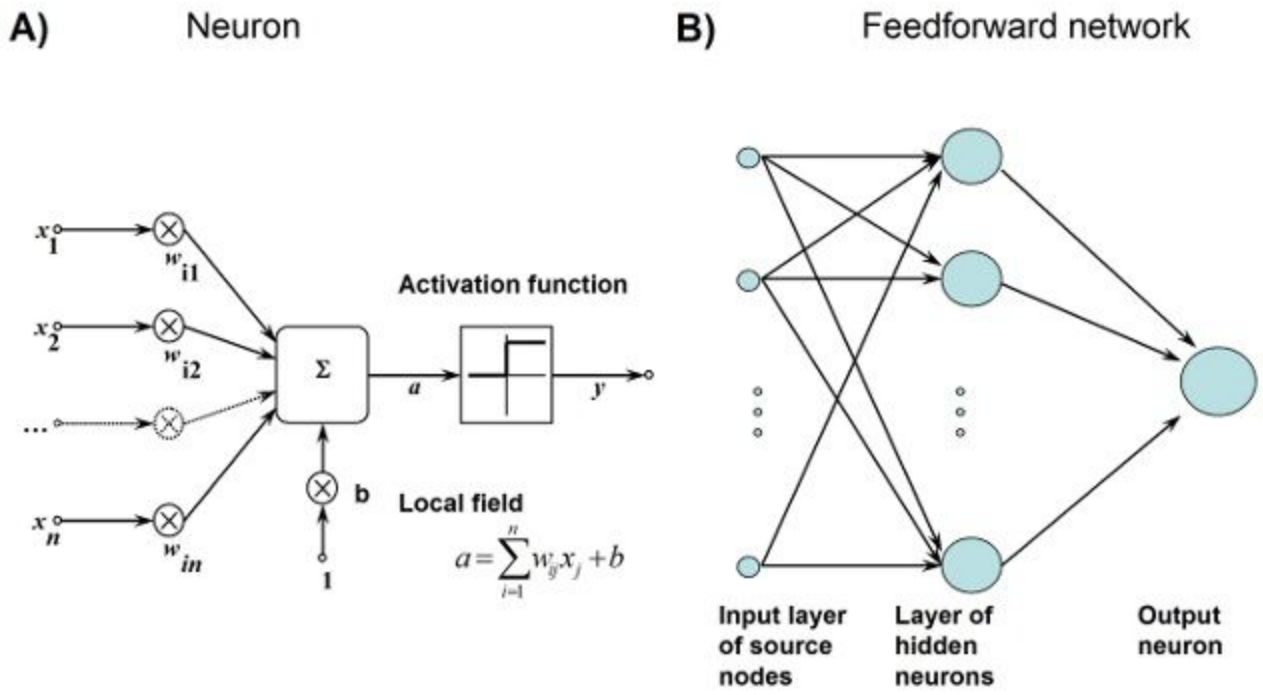
Now, the magic here is that we give a few input - output pairs to the network, so that it learns the occurring patterns of hidden neurons and synapses, and finally can determine the output for any given input based on this learning.

HOW DOES A NEURAL NETWORK WORK?

A neural networks consists of at least two layers viz. input and output. Additionally, there can be many hidden layers to study the nonlinear relationships among inputs. Every neuron has a parametric value which is passed through an activation function to decide whether to consider that neuron for output building, or not. Once a neuron passes the activation test, its new value is multiplied by a weight which decides how much of it is to be taken into the output. At the output node, all such values from all the neurons are then added and passed to activation function to give the value for this output neuron. This is called forward propagation



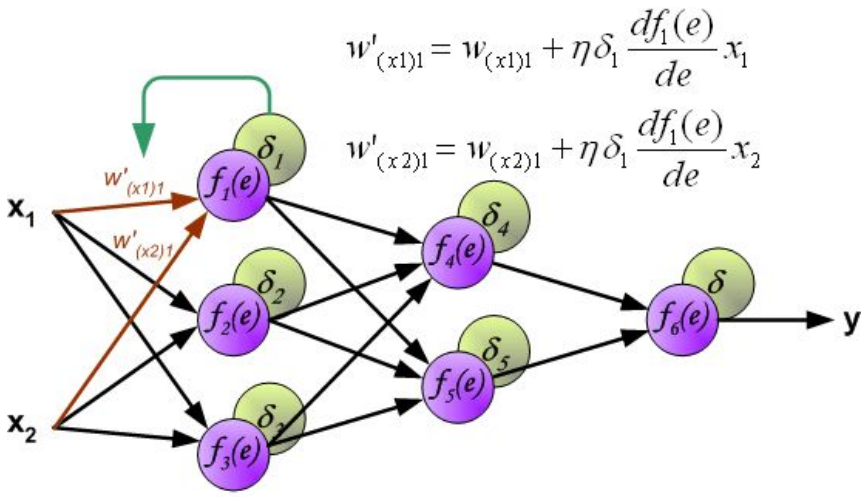
In the training phase, we compare this generated output with the actual output, and the error is studied. To reduce the error, we now need to supervise the amounts by which each neuron was taken into account, or alter the weights / synapse.



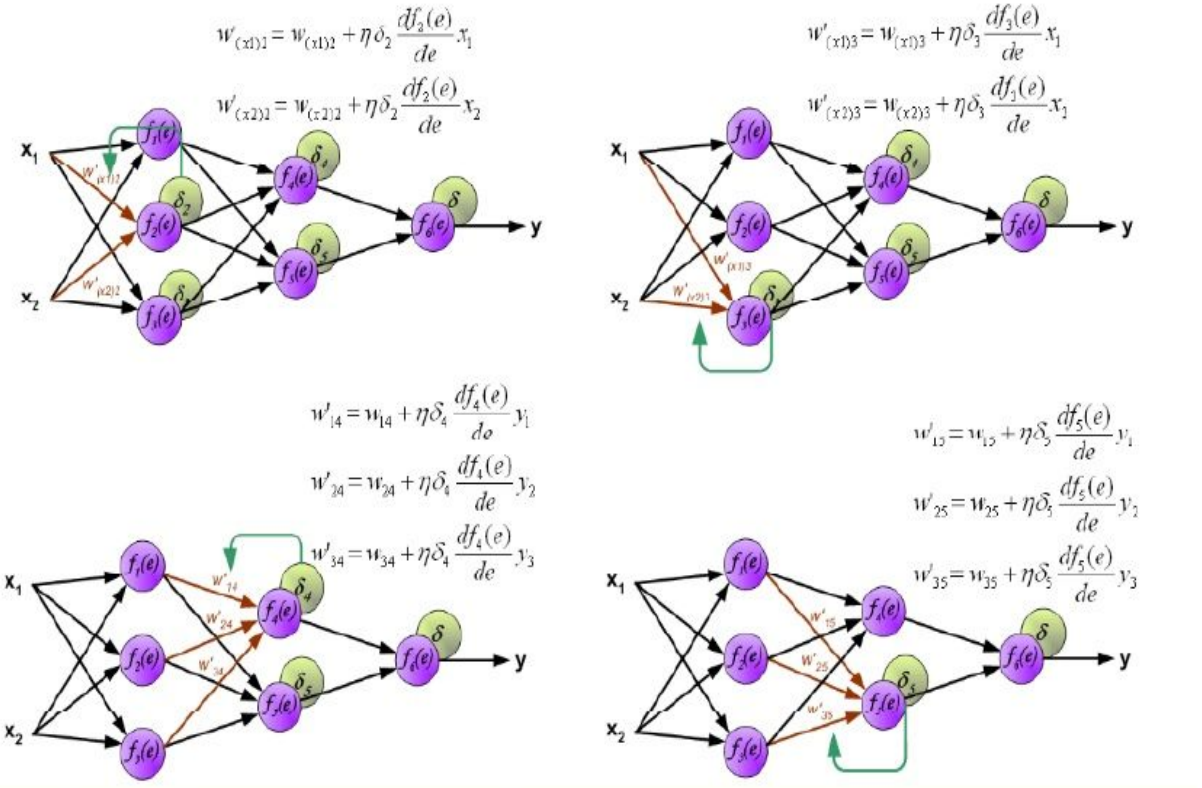
But how?

This is where the magic comes. This is done by backward propagation. It also works layer wise, starting from output layer to the hidden layer.

For example, if we need to do backward propagation from layer 1 to layer 0 for neuron x_1, following algorithm is followed.



Following the same pattern over the whole graph,



For more explanations, please watch these video lectures.

HOW DID WE IMPLEMENT IT?

Well, the mathematical functions of neural network seem quite tough to understand. But python implementation is relatively easy.
We have referred the neural network implementation given in the this github repository.
And this is our attempt of plugging the above stated neural network into the problem of a support bot.

As described here, we are currently working with a 3 layer neural network :

1. Input layer : Number of neurons in this layer is equal to the number of parameters we wish to take into account for all the posts.
2. Hidden Layer : We have taken the number of neurons equal to 3 here.
3. Output Layer : Our only output is the fitness value of the corresponding answer. Hence there is only one neuron here.

Firstly, we generate the input and the output vector.

```
def gen_ip_op():
    max_link, max_word, max_like, max_read = find_max()

    x = []
    y = []
    name1 = []

    out = frappe.db.sql("""SELECT name FROM tabposts WHERE fitness > 0.0 AND from_neural = 0;""", as_dict=True,
         debug=True)

    i=0
    for row in out:
        x_in, fit_x, no_parameters, parameter_list = gen_vector(row.name, max_word, max_link, max_like,
            max_read)
        name1.append(row.name)
        x.append(x_in)
        y.append(fit_x)
        i = i+1

    print "Max answers : ", i
    X = np.array(x)
    Y = np.array([y])
    name = np.array([name1])
    Y = Y.T
    name = name.T
    return (X, Y, name, no_parameters, parameter_list, i, max_link, max_word, max_like, max_read)
```

Two dimensional input matrix is created here, where each row corresponds to the parameters of each trained post. This row vector is generated in the following function.

```
def gen_vector(name, max_word, max_link, max_like, max_read):
    doc = frappe.get_doc('posts', name)

    word = float(doc.word_count)/max_word
    like = float(doc.like_score)/max_like
    read = float(doc.reads)/max_read
    link = float(doc.incoming_link_count)/max_link
    fit_x = float(doc.fitness)
    isdoc = float(doc.is_doc)
    avg_word = float(doc.avg_word)
    avg_sentence = float(doc.avg_sentence)
    if(doc.reply_to_post_number):
        reply = float(1)
    else:
        reply = float(0)
    no_parameters = 3
    parameter_list = ["word", "avg_word", "avg_sentence"]
    parameter_list = json.dumps(parameter_list)
    x_in = np.array([word, avg_word, avg_sentence])
    return (x_in, fit_x, no_parameters, parameter_list)
```

Maximum values for each parameter are required to normalize the values. They are generated here.

```
#find max of each parameter
def find_max():
    max_link = max_word = max_like = max_read = 1

    rows = frappe.db.sql("""SELECT incoming_link_count, `reads`, like_score, word_count FROM tabposts;""",
        as_dict=True, debug=True)

    for row in rows:
        if row.incoming_link_count > max_link:
            max_link = row.incoming_link_count
        if row.word_count > max_word:
            max_word = row.word_count
        if row.like_score > max_like:
            max_like = row.like_score
        if row.reads > max_read:
            max_read = row.reads

    return (max_link, max_word, max_like, max_read)
```

Now, these vectors are fed to the neural network in the following function, and the records are stored in the DocType ‘neural_net_traces’.

```
def neural_network():

    X, Y, name, no_parameters, parameter_list, total_answers, max_link, max_word, max_like, max_read
        gen_ip_op()


    np.random.seed()

    NN = Neural_Network(no_parameters)

    T = trainer(NN)
    T.train(X, Y)

    err = np.mean(np.abs(NN.error))

    calculate_fitness(NN.W1, NN.W2, max_link, max_word, max_like, max_read)

    _syn1 = json.dumps(NN.W1.tolist())
    _syn2 = json.dumps(NN.W2.tolist())

    frappe.get_doc({
                'doctype': 'neural_net_traces',
                'tot_answers': total_answers,
                'params': parameter_list,
                'no_params': no_parameters,
                'hidden_size': NN.hiddenLayerSize,
                'syn1': _syn1,
                'syn2': _syn2,
                'error': err
            }).insert()
    frappe.db.commit()

    return
```

calculate_fitness() function is used to implement the learning of the neural network into the calculation on the fitness values of other untrained answers.

```
def calculate_fitness(syn1, syn2, max_link, max_word, max_like, max_read):
    rows = frappe.db.sql("""SELECT name FROM tabposts WHERE fitness = 0.0""", as_dict=True, debug=True)

    for row in rows:
        doc = frappe.get_doc('posts', row.name)
        print doc.name
        x_in, fit_x, no_parameters, parameter_list = gen_vector(row.name, max_word, max_link, max_like,
            max_read)
        print x_in
        hidden = sigmoid(np.dot(x_in, syn1))
        print hidden
        x_out = sigmoid(np.dot(hidden, syn2))
        print x_out
        doc.fitness = float(x_out[0])
        doc.from_neural =  1
        doc.save(ignore_permissions=True)
        frappe.db.commit()

    return
```

EXPERIMENTS CONDUCTED:

Here, after running the neural network a few times with different combinations of parameters, we observed that the selection of parameters and their order in the input vector plays an important role in the overall mean error.

For example, for our database, when the number and type of parameters considered for the formation of input vector were changed, the mean error varied as observed below

Input Vector	Mean Error
["word", "avg_word", "avg_sentence"]	0.345
["word", "avg_word", "avg_sentence", "isdoc"]	0.154
["avg_sentence", "avg_word", "word", "like"]	0.187

Also, for the same group of parameters, when their ordering in the input vector was altered, following changes were observed in the mean error

Input Vector	Mean Error
["word", "avg_word", "avg_sentence"]	0.345
["word", "avg_sentence", "avg_word"]	0.183
["avg_word", "avg_sentence", "word"]	0.184
["avg_sentence", "avg_word", "word"]	0.203

Now, we made a third experiment by changing the number of neurons in the hidden layer. Putting the input vector to be constant as ["word", "avg_sentence", "avg_word"], we observed the following pattern

Size of Hidden Layer	Mean Error
1	0.259
2	0.259
3	0.183
4	0.203
5	0.145
6	0.203
7	0.185
8	0.183
9	0.174
10	0.186

Moreover, we can also alter the python code to implement more than one hidden layer and hence vary the number of neurons in both of them, and thus obtain a suitable combination with least mean error.

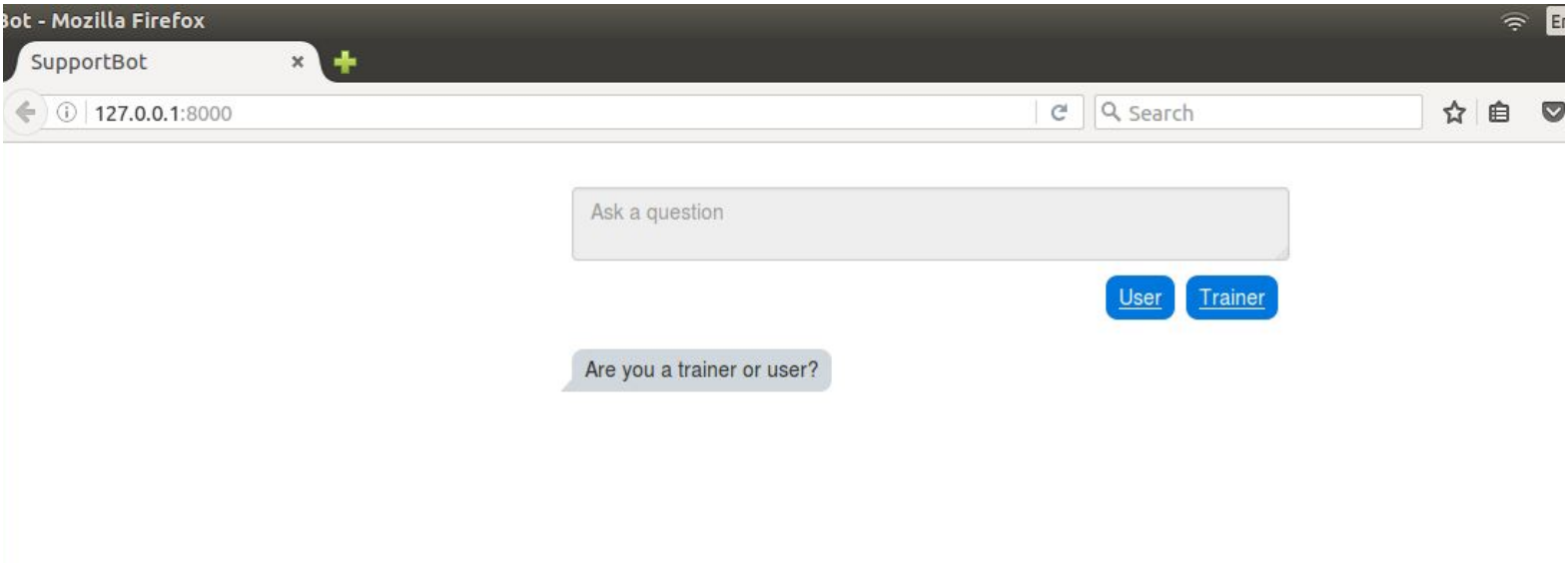
Thus, in order to reduce the mean error, you must experiment with many different permutations and combinations of the input parameters, and different number of neurons in the hidden layers. It is safe to say that such a variation will be unique for all databases.

For the best output, select a variant of input parameter which gives minimum error, and hence calculate the fitness values of all the untrained answers.

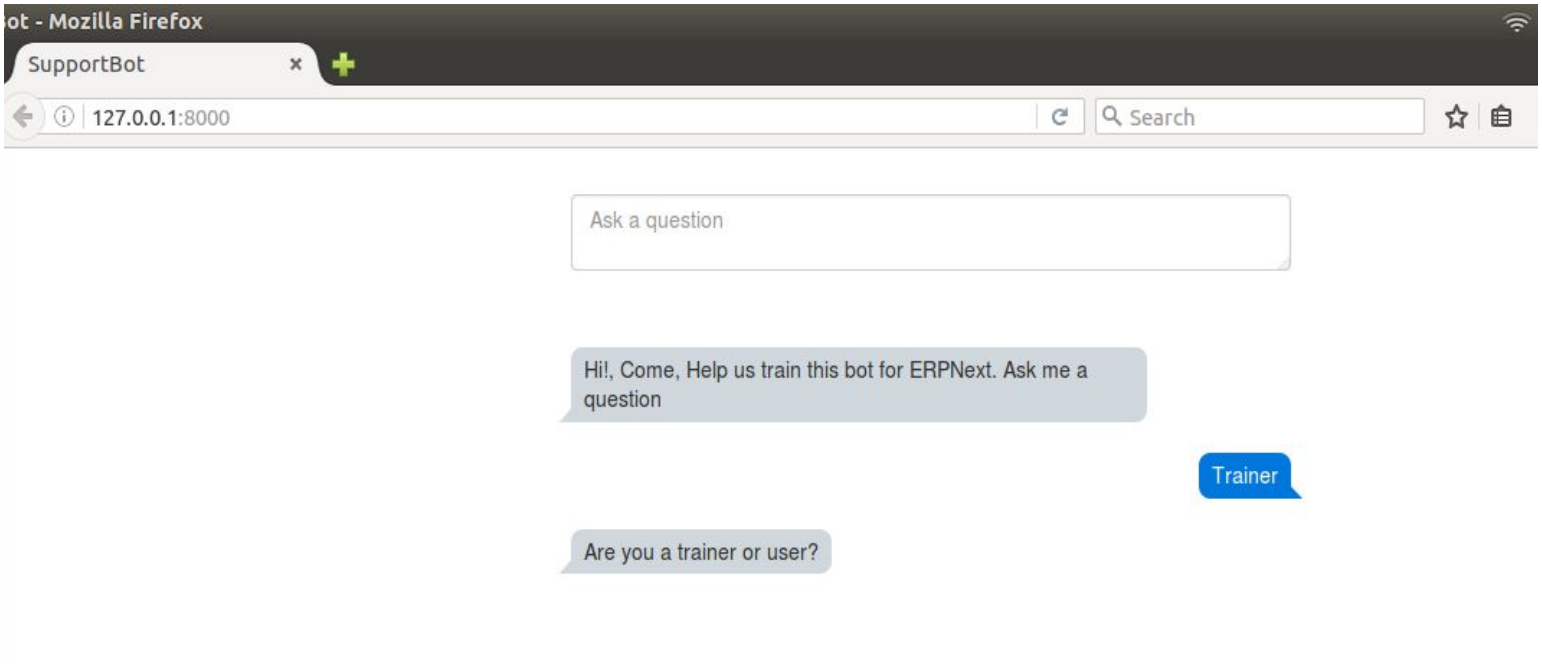
TRAINING PHASE:

Now, once the algorithm has been put, the bot needs to be trained. This is better done by experts from customer support department, but they hardly know how to handle a backend script. Thus, the first thing to do was to develop a user interface.

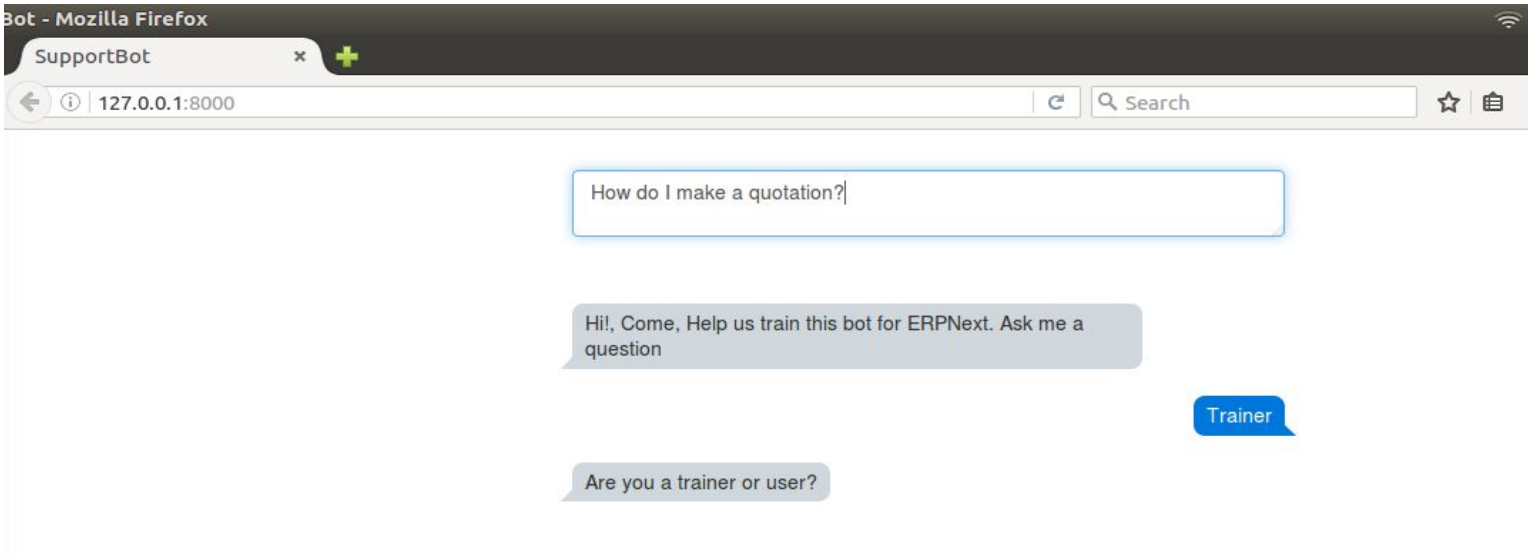
We developed a user interface with two modes - User and Trainer. User mode is to be used for production phase, and Trainer mode is to be used for Training phase.



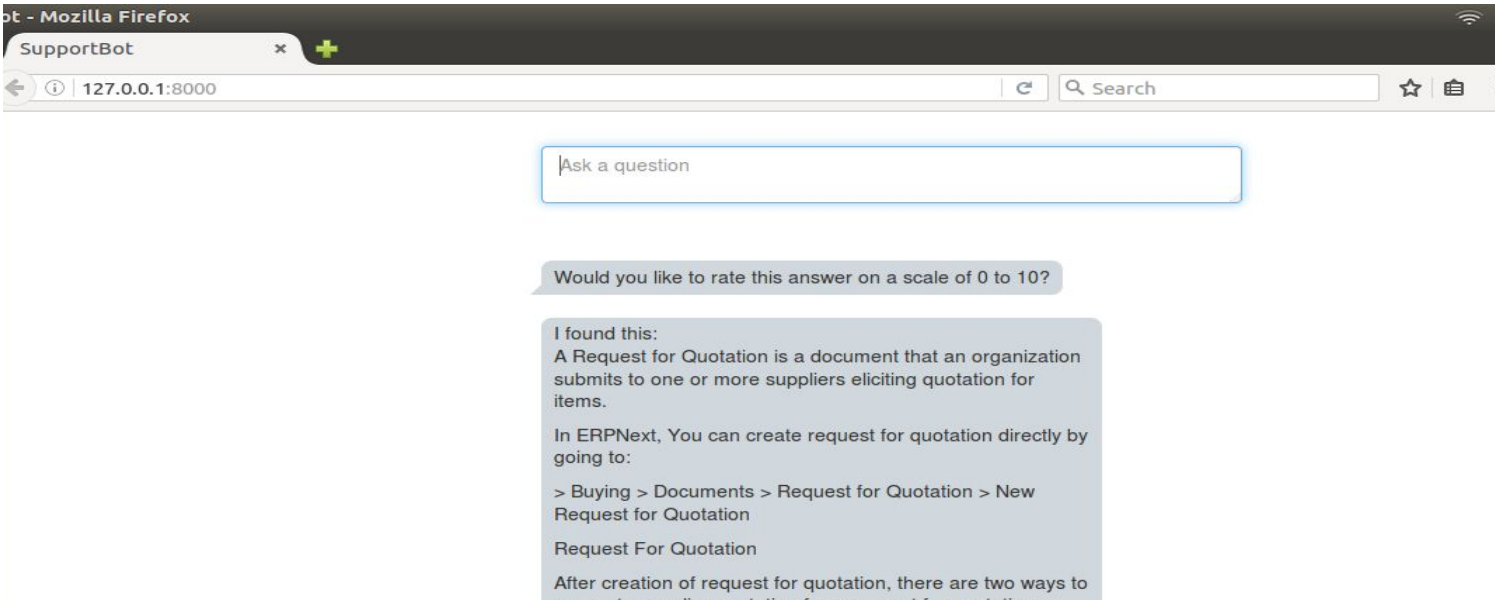
First, Let us see the Trainer Mode.



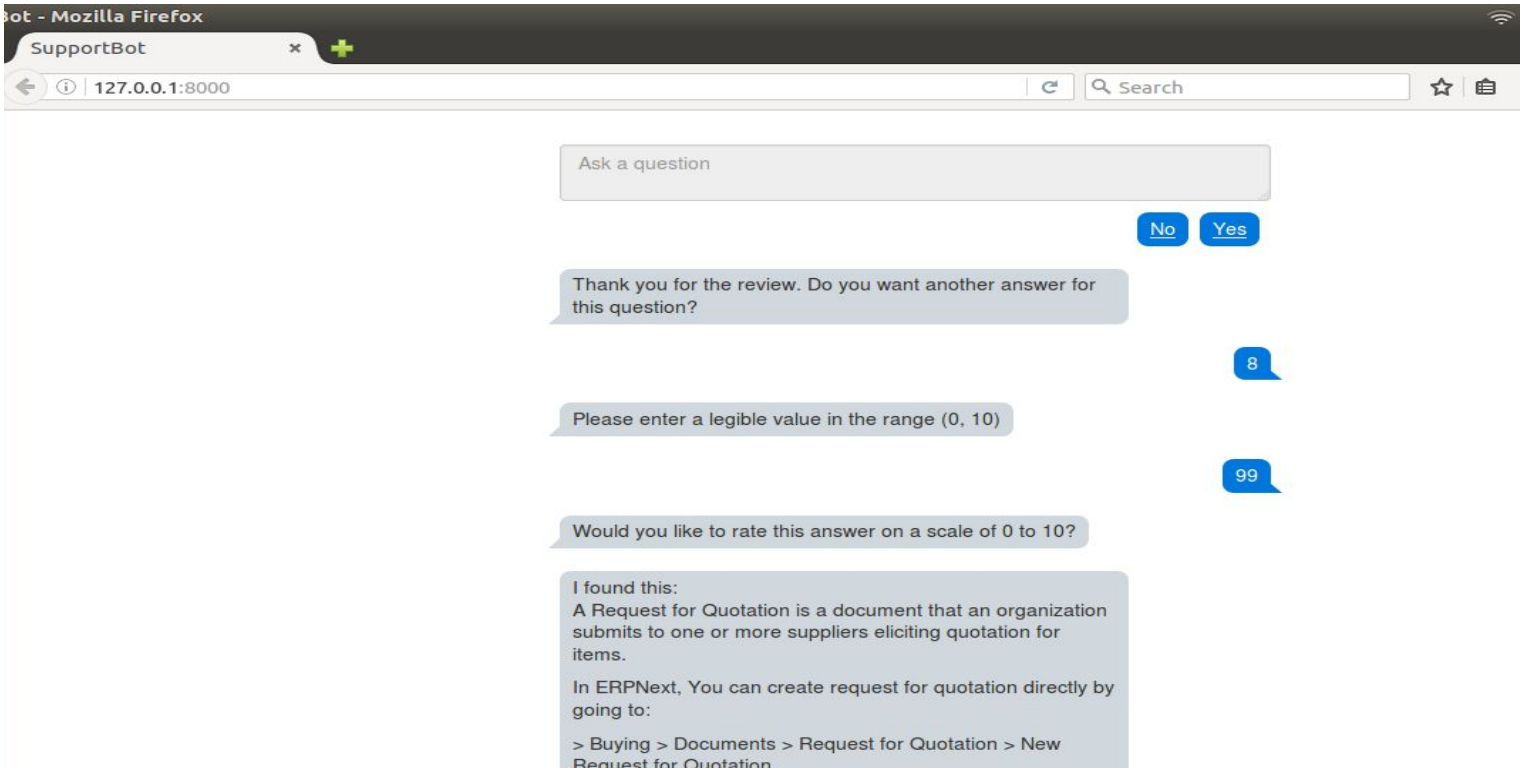
In the trainer mode, you get to be the customer support expert who can rate the answers and help us train our bot. Once you choose 'Trainer' option, enter your question in the text box, and press enter key.



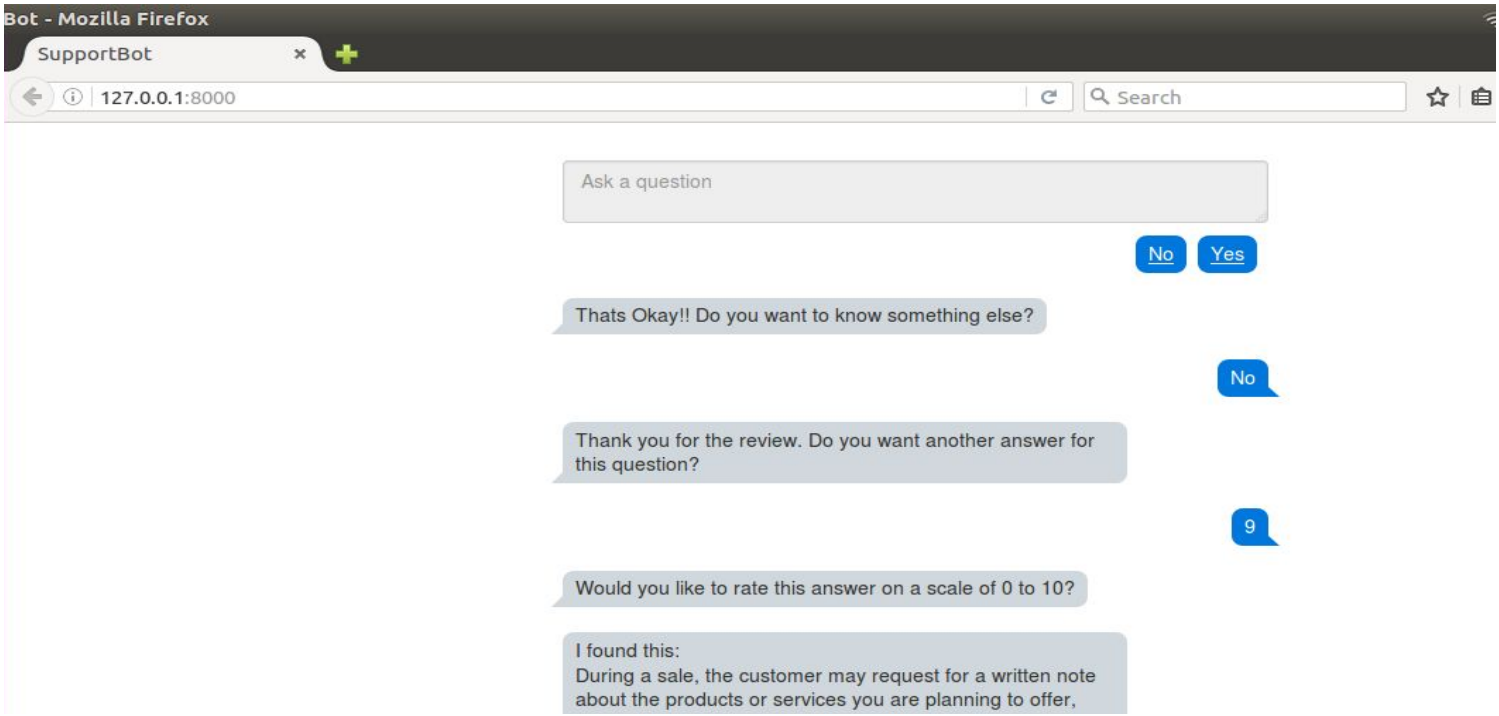
This will fetch you an answer from the database (composed of mainly the documentations, but can be extended) based upon the Full Text Search of MySQL.



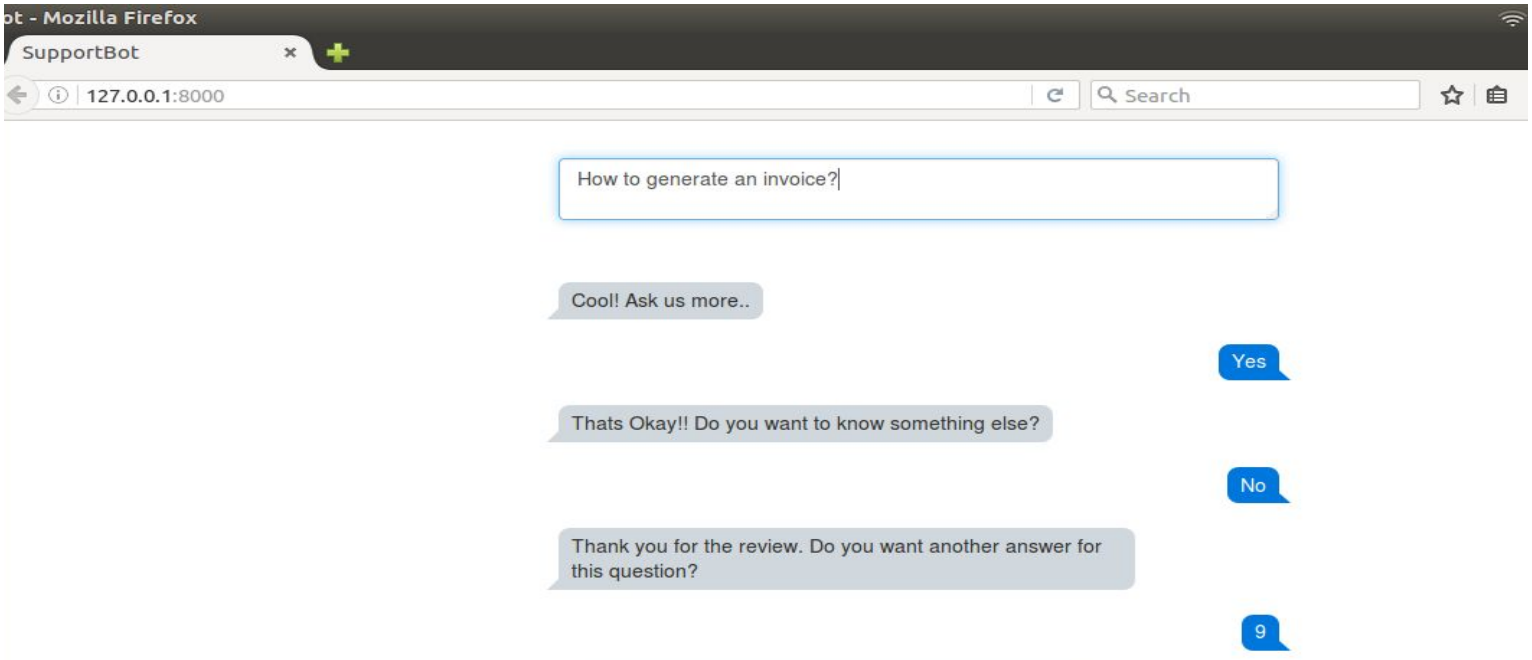
You will be prompted to enter a fitness value score (between 0-10) for this answer with relevancy to your question. (If you do not wish to rate the answer, simply press the enter key). If you enter a value beyond the acceptable range, an error message will be displayed and you will have to enter a legible value.



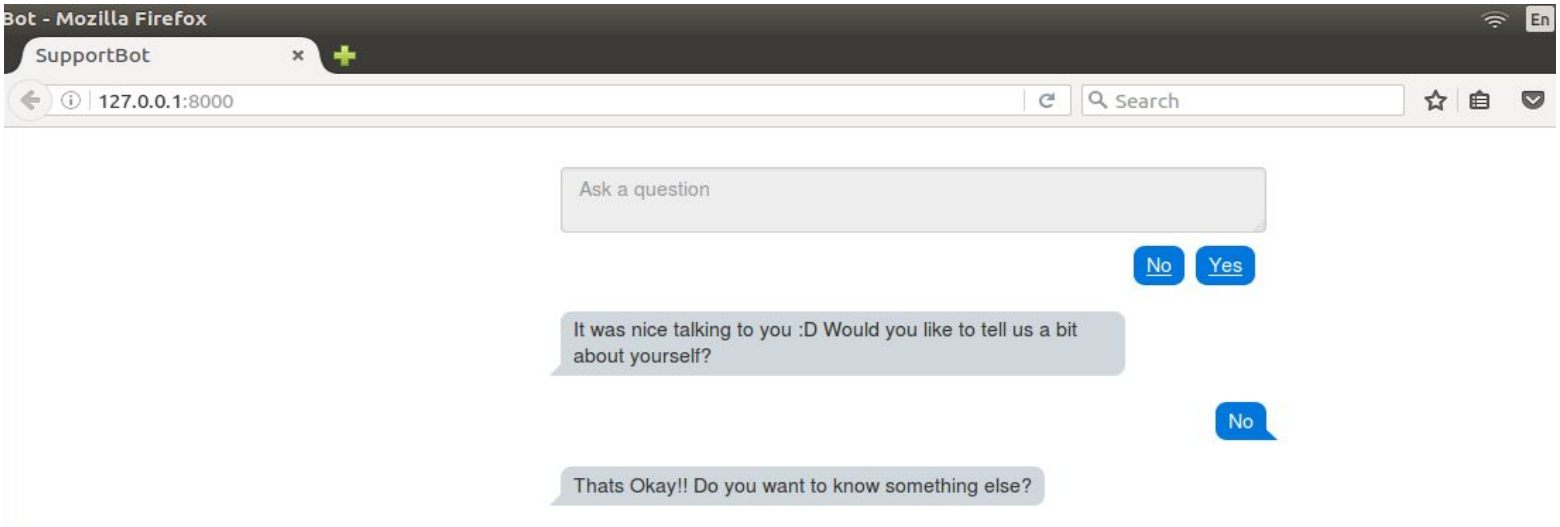
Once you give a legible fitness score, you may or may not select to view another answer.
If you click 'Yes', another answer will be displayed and you will be asked to rate it. If you choose 'No', following happens.



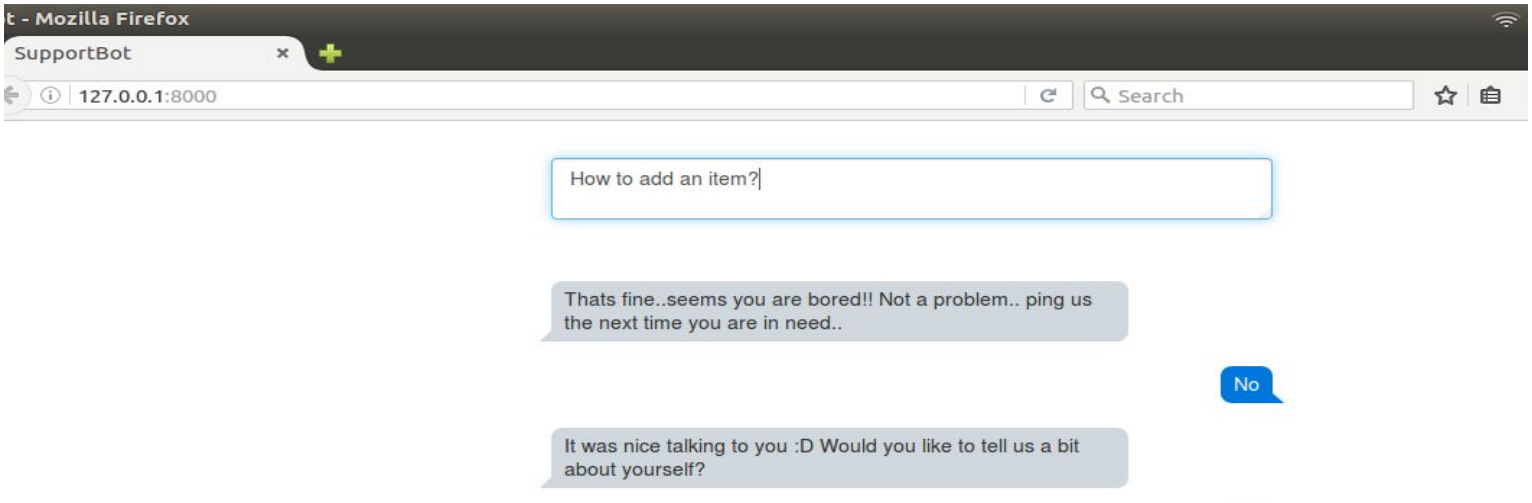
Now, if you wish to ask a new questions, choose ‘Yes’, and enter your next question in the text box.



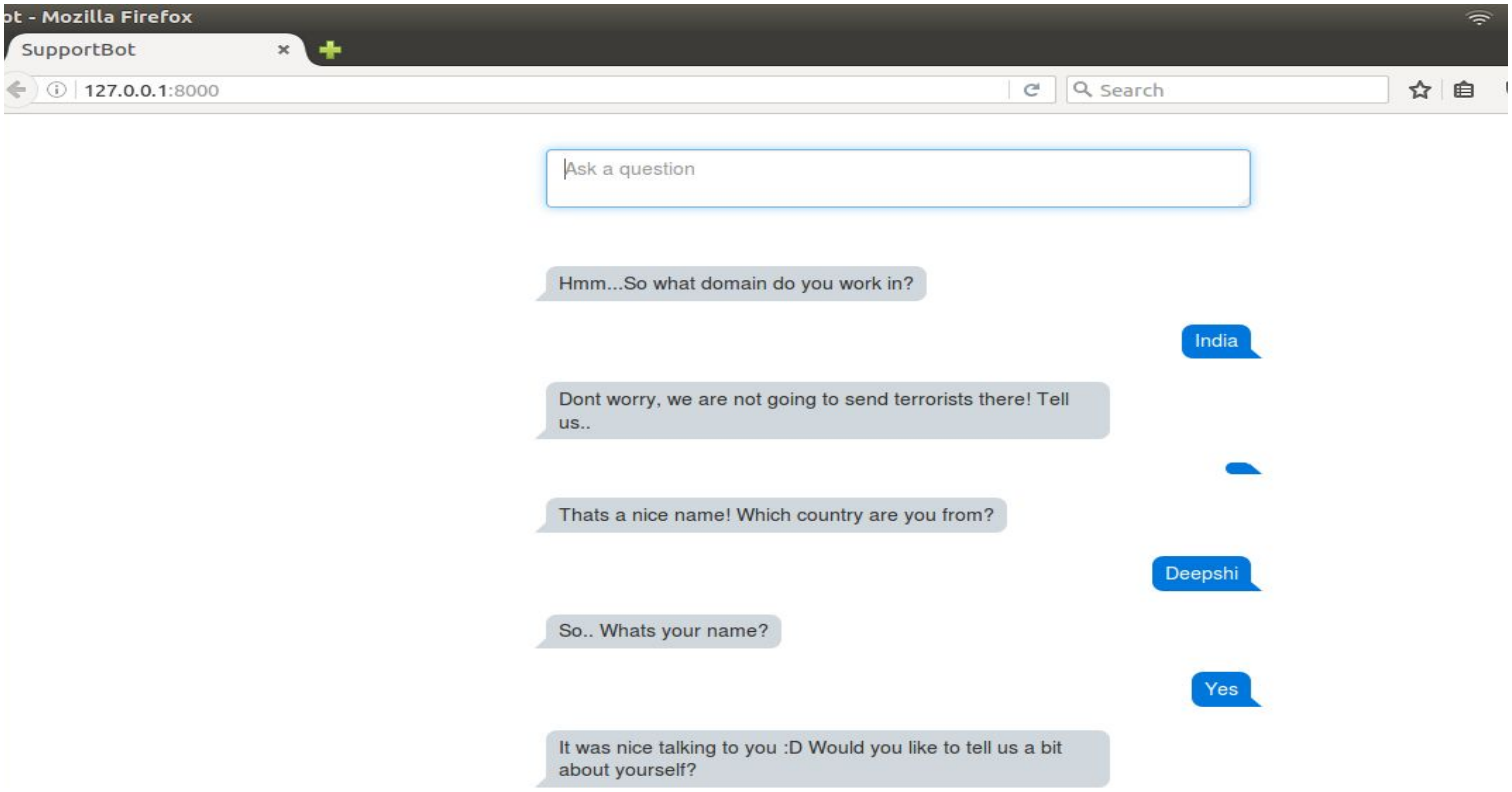
If you choose ‘No’, you will enter the ‘introduction’ mode, where you may choose to leave your feedback with us

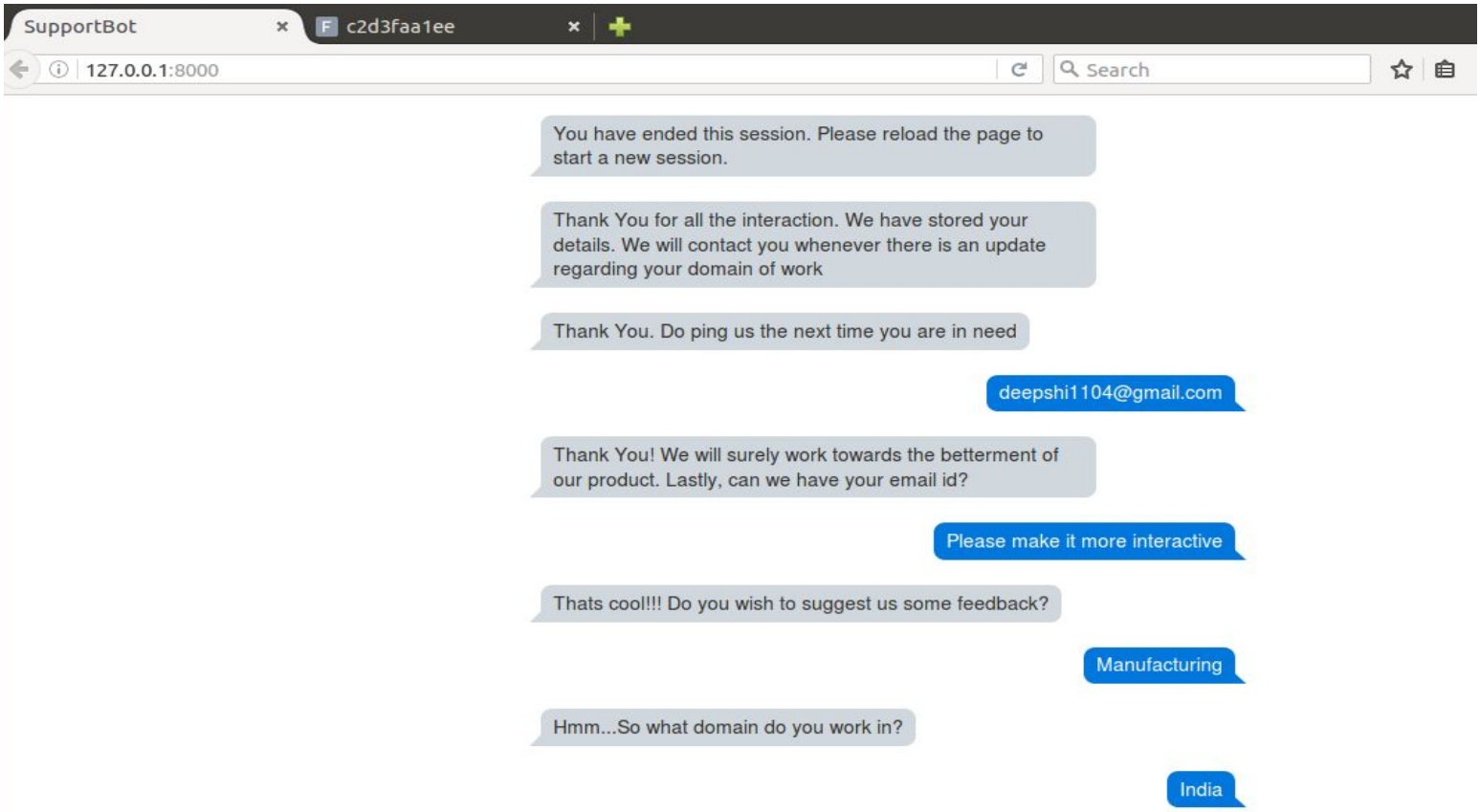


If you choose ‘No’, you may continue as before, by putting up a new question in the text box, and the same process repeats.

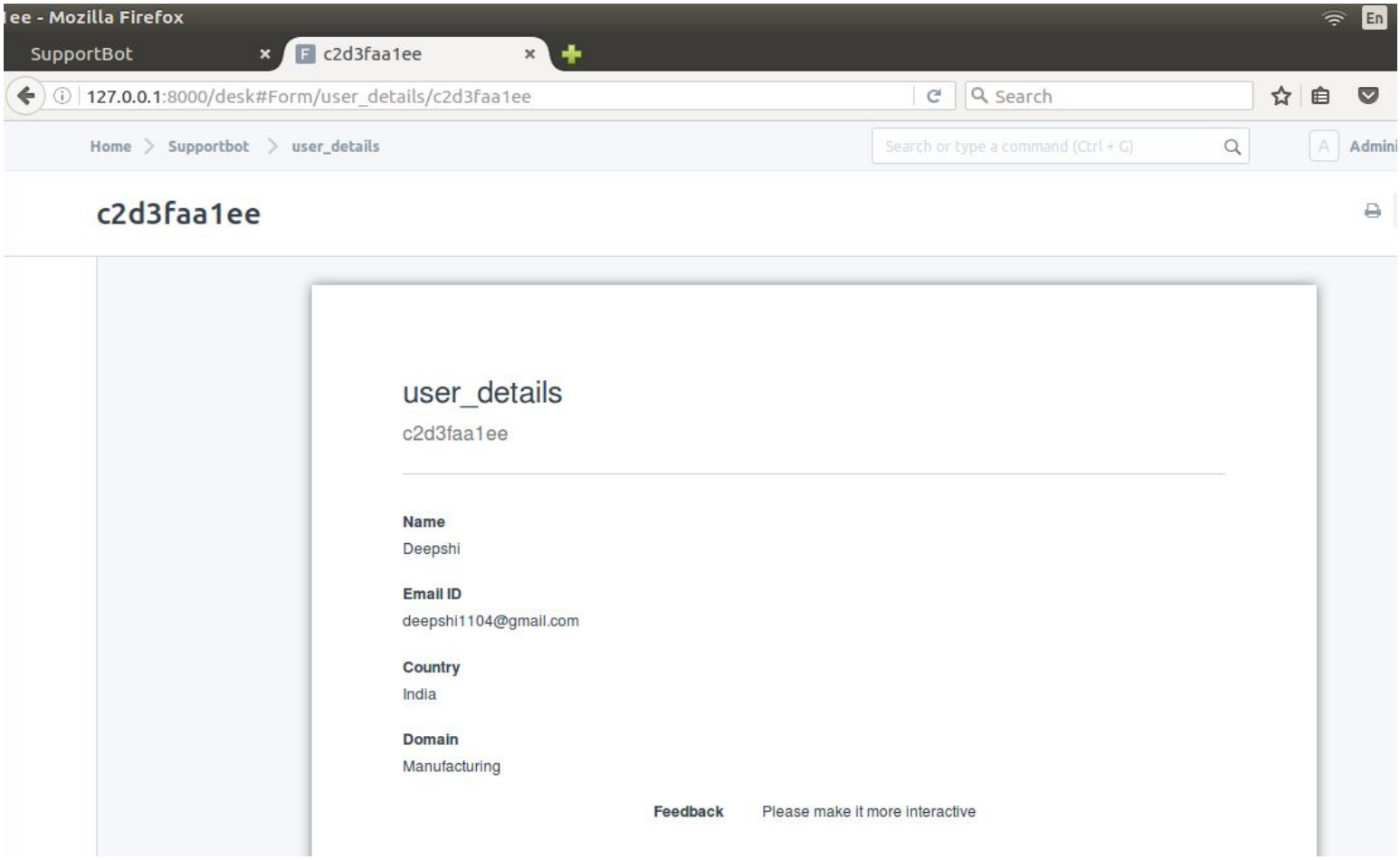


If you choose 'Yes', you will be prompted to share your details and leave a feedback. Appropriate error handling is done in a way such that no field can be left empty.



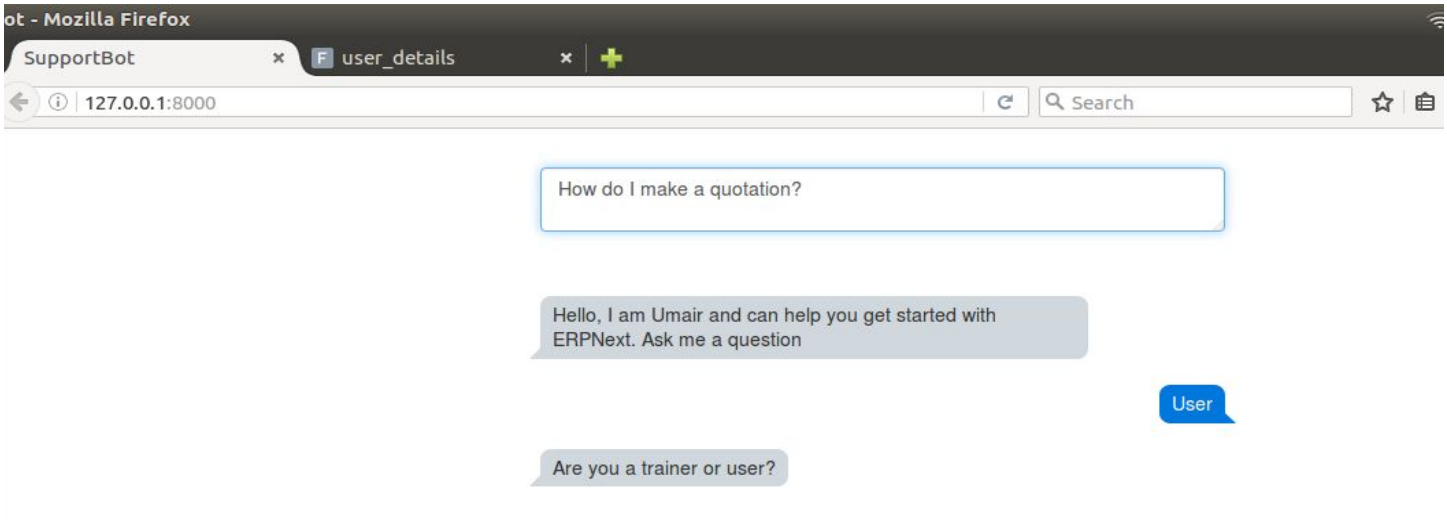


You details will be stored in the ‘user_details’ doctype.

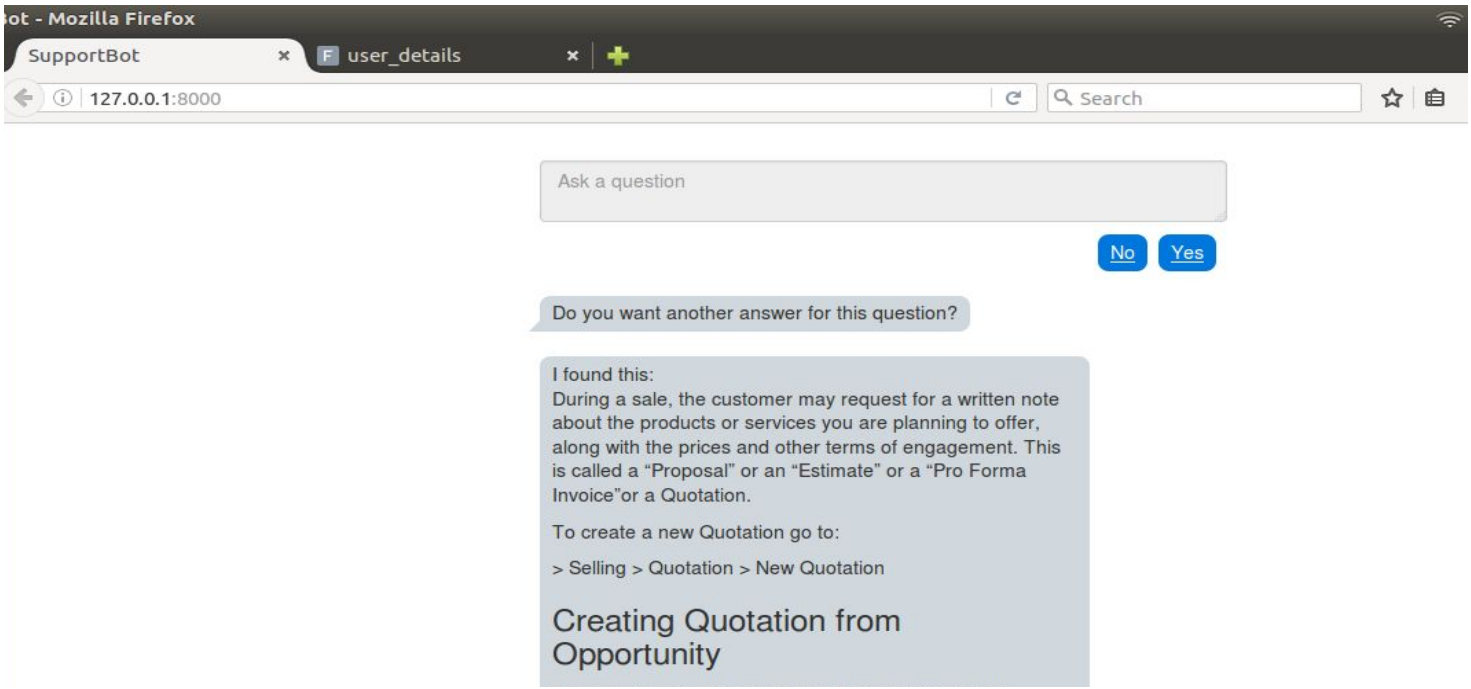


PRODUCTION PHASE :

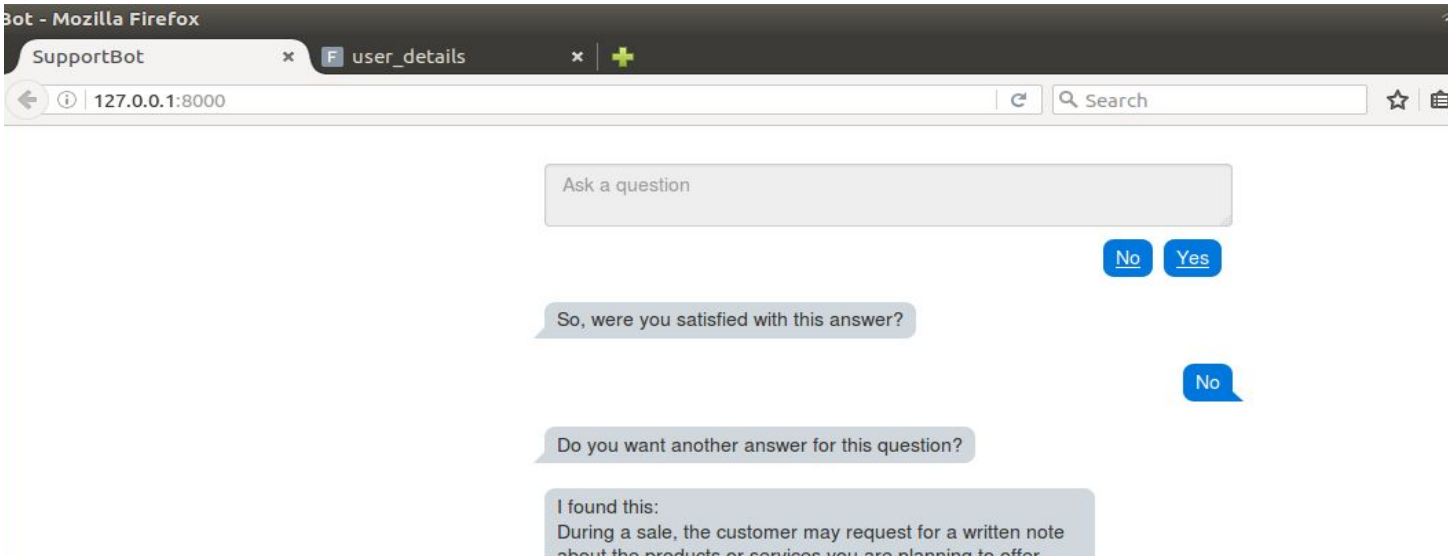
Here, the User mode form the user interface is used. If you wish to use the bot like a normal user, and not rate the answers, then choose ‘User’, and enter your question in the text box.



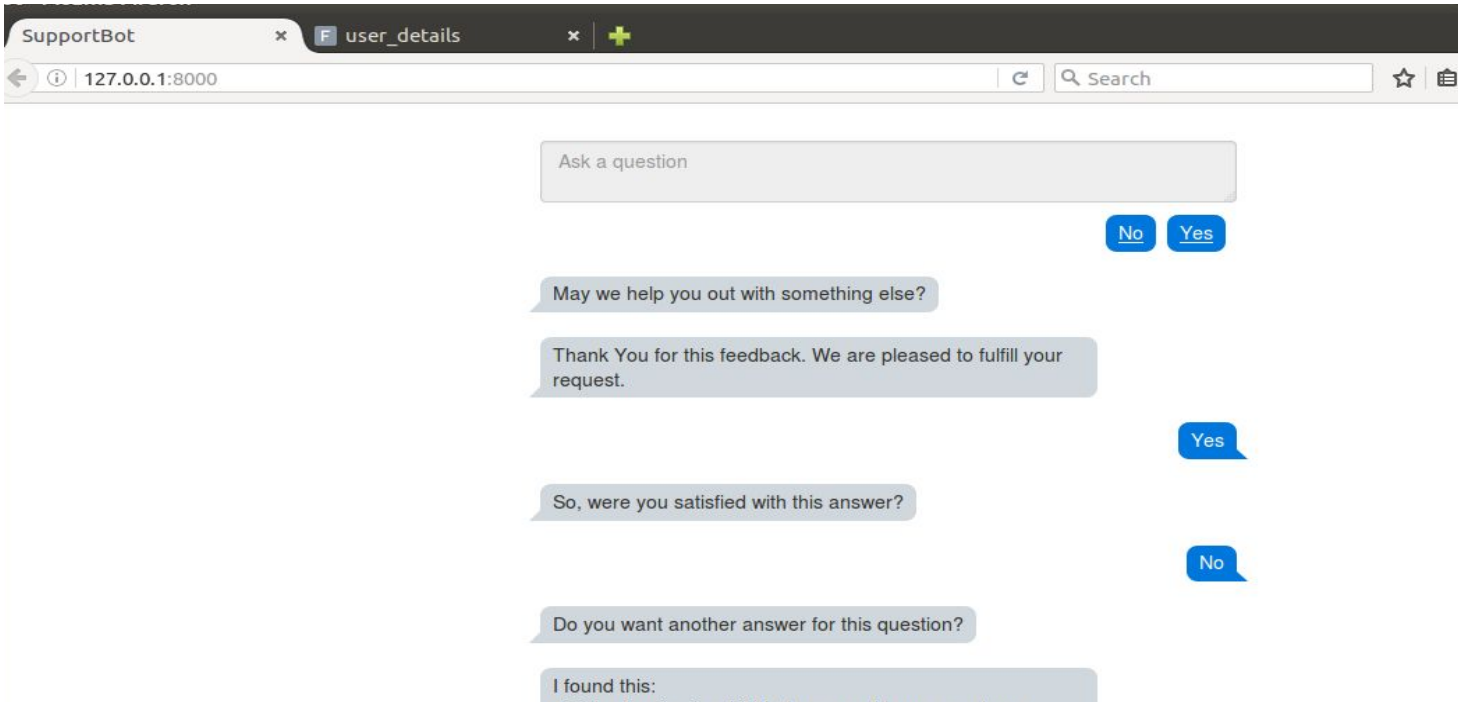
On receiving a question, the bot selects top 15 answers from the database in order of the fitness value as well as the score given by the full text search of MySQL. The fitness values here (for the answers not trained by an expert) are calculated by the neural network. Once, the sequence of suitable answers has been generated, they are displayed to the user accordingly one by one.



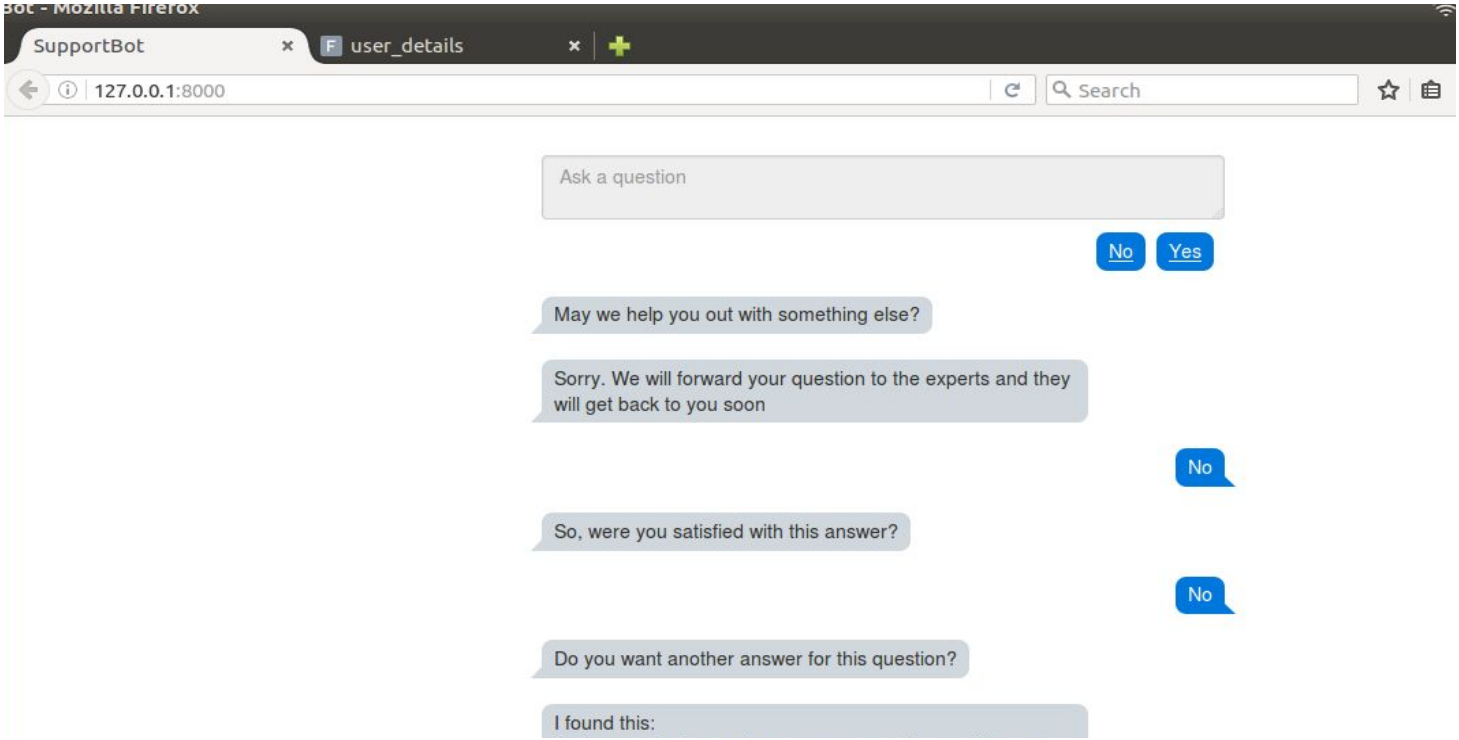
If you choose ‘Yes’, next answer in the sequence will be displayed. If you choose a ‘No’, your feedback is asked for.

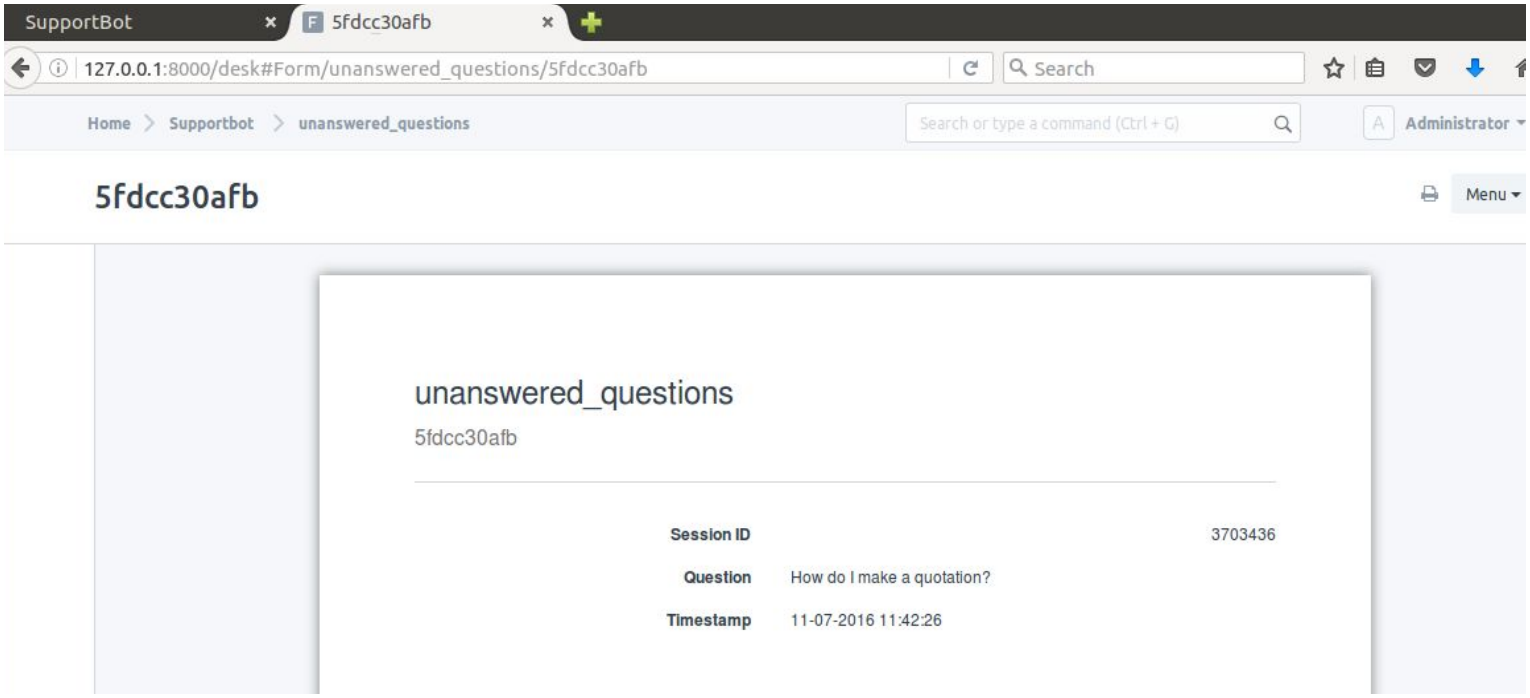


Here, If you were satisfied with the answer, the bot will interpret as you like the answer, and the number of likes of this answer would be incremented by one.



If you were not satisfied with the answer, choose a ‘No’. This would mean that our current database is not sufficient to answer your question, and hence it would be stored in the DocType ‘unanswered_questions’ and will be seen by the experts.


SupportBot
user_details
127.0.0.1:8000
Search
Ask a question
No
Yes
May we help you out with something else?
Sorry. We will forward your question to the experts and they will get back to you soon
No
So, were you satisfied with this answer?
No
Do you want another answer for this question?
I found this:


SupportBot
5fdcc30afb
127.0.0.1:8000/desk#Form/unanswered_questions/5fdcc30afb
Search
Home
Supportbot
unanswered_questions
Search or type a command (Ctrl + G)
Administrator
5fdcc30afb
Menu
unanswered_questions
5fdcc30afb
Session ID
3703436
Question
How do I make a quotation?
Timestamp
11-07-2016 11:42:26

Now, for the question ‘May we help you out with something else?’, you may choose ‘Yes’ if you wish to ask another question. Choose a ‘No’ if you are done with your queries.

On choosing a ‘No’, you will be asked to share your details and feedback. You may opt to choose a ‘No’ here too.

EXTRA FEATURES:

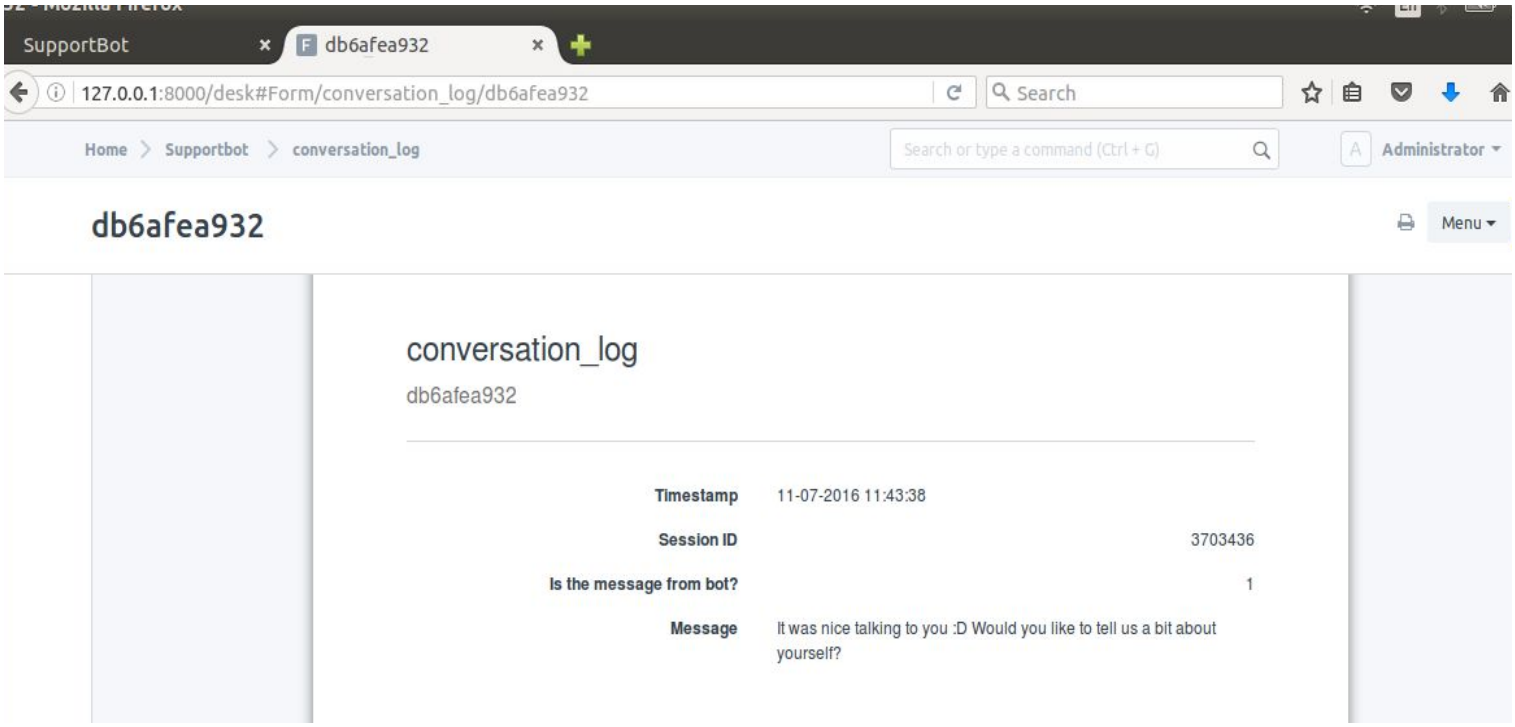
You may login to the desk with user id : Administrator, password : qwe

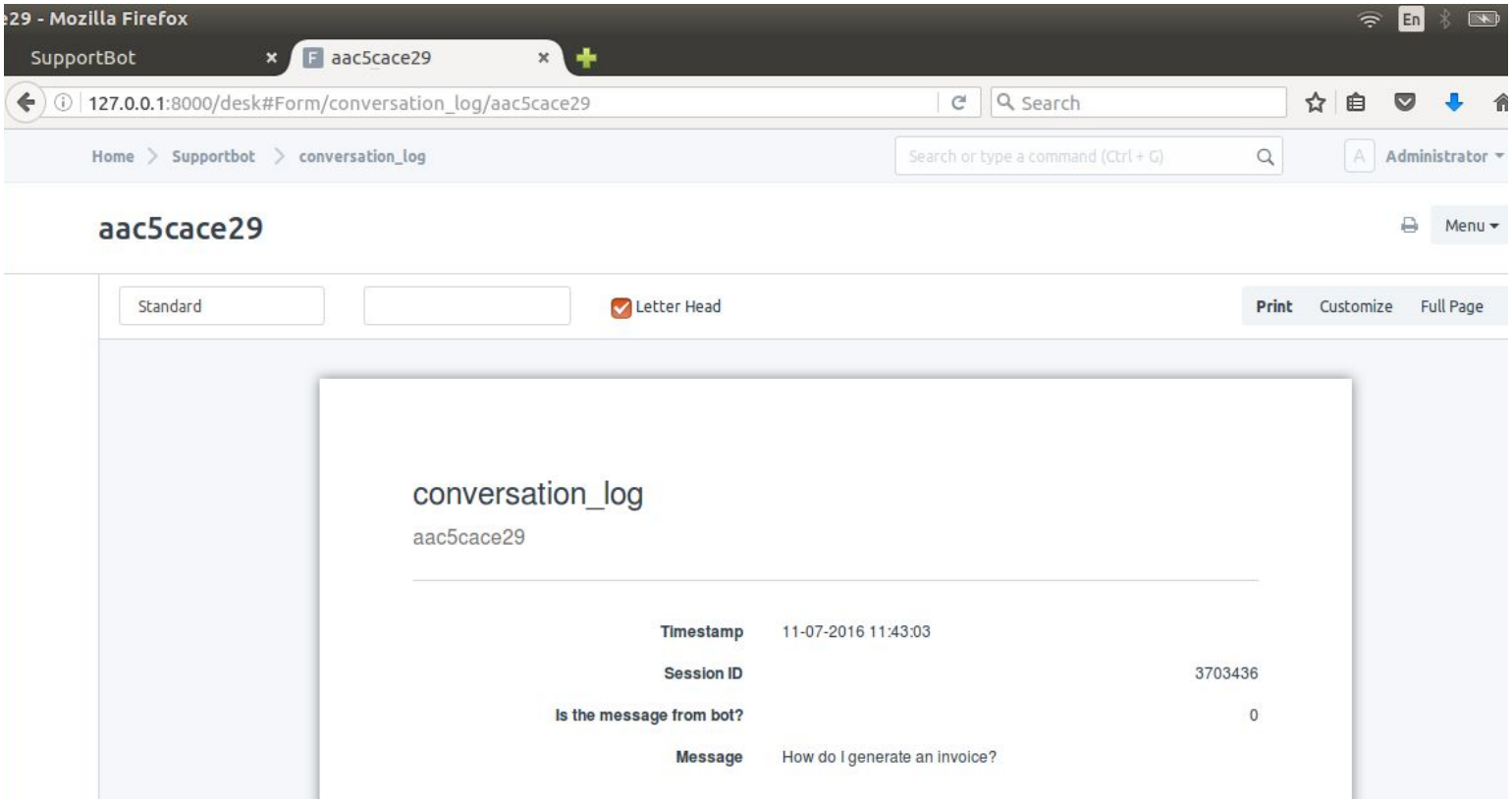
We have implemented the following noteworthy features here:

1. User Details
2. Conversation Logs
3. Neural Net Traces
4. Feedback
5. Unanswered Questions

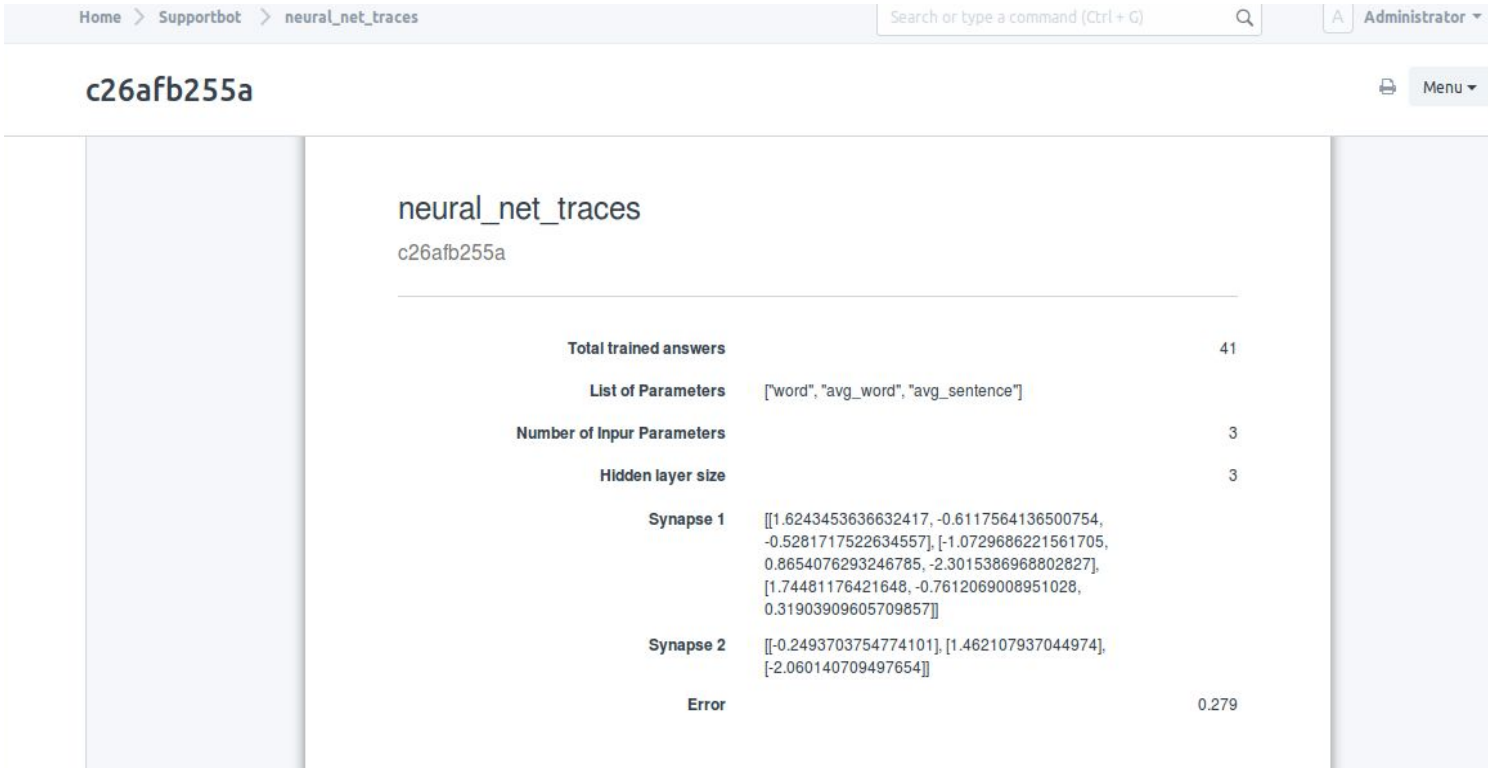
User Details stores the details of users in both Trainer and User mode as described above

Conversation Logs stores the details of all the conversations taking place between the bot and the user only in User mode.





Neural Net Traces stores the details of the neural network, every time a training algorithm is run for it.



Feedback : When in User modes, the user is satisfied with the answer, the like score of the answer is incremented by one.

```
@frappe.whitelist(allow_guest=True)
def increment_like(name):
    doc = frappe.get_doc('posts', name)
    doc.like_score = doc.like_score + 1
    doc.save(ignore_permissions=True)
    frappe.db.commit()
    return
```

Unanswered Questions : When in User mode, the user is not satisfied with any of the answers displayed, the question is stored in the DocType ‘unanswered_questions’ as described earlier.