CHATBOT USING PYTHON

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Prabhu M(RA2011003010827)
Kongarasan S (RA2011003010856)
Vickram M (RA2011003010873)

Under the guidance of

Dr. M. Rajalakshmi

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **“CHATBOT USING PYTHON”** is the bona fide work of **PRABHU M(RA2011003010827), KONGARASAN S(RA2011003010856), VICKRAM M(RA2011003010873).** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. M. Rajalakshmi

Assistant Professor
Department of Computing Technologies

SIGNATURE

Dr. M. Pushpalatha
HEAD OF THE DEPARTMENT
Professor & Head
Department of Computing Technologies

ABSTRACT

This report presents the development of a chatbot using Python Flask, a web framework for building web applications. The chatbot is designed to assist users in a range of tasks, such as answering frequently asked questions and providing customer support. The development process involved several steps, including defining the chatbot's purpose, selecting an appropriate natural language processing (NLP) model, and implementing the chatbot using Flask.

To improve the chatbot's functionality, we used the Natural Language Toolkit (NLTK) to preprocess user input and extract relevant information. We also used a pre-trained NLP model to analyze user input and generate appropriate responses. Through user testing, we evaluated the chatbot's performance and identified areas for improvement.

The chatbot's implementation using Flask allowed us to create a user-friendly web interface that integrates seamlessly with other web applications. We also used Flask's modular design to create a scalable architecture that can be easily extended with additional features.

Overall, the chatbot's development using Python Flask demonstrates the potential of chatbots in various industries, such as customer service and e-commerce. With further development and refinement, chatbots can provide significant benefits to both businesses and customers, including increased efficiency and improved user experience.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IOT	Internet of Things
PIR	Passive Infrared
LCD	Liquid Crystal Diode
DHT	Distributed hash table
IR	Infra red
UART	Universal Asynchronous Receiver/Transmitter
IDE	Integrated Development Environment

CHAPTER 1

INTRODUCTION

The development of chatbots has gained significant traction in recent years, with many businesses leveraging this technology to improve customer engagement and streamline operations. Chatbots are computer programs that simulate conversation with human users, and they are capable of answering questions, providing recommendations, and even completing transactions. Python Flask, a popular web framework, has emerged as a powerful tool for building chatbots due to its flexibility, scalability, and ease of use.

In this report, we present the development of a chatbot using Python Flask. Our chatbot is designed to assist users with a range of tasks, including answering frequently asked questions and providing customer support. To develop the chatbot, we utilized natural language processing (NLP) techniques to improve its functionality and provide a seamless user experience.

The report outlines the development process, including the selection of an appropriate NLP model, the integration of Flask for web development, and the evaluation of the chatbot's performance through user testing. We also discuss the potential benefits of chatbots for businesses, including increased efficiency, improved customer engagement, and reduced costs.

As the use of chatbots continues to grow, it is essential to have a deep understanding of the underlying technology and development frameworks. This report provides a comprehensive guide to developing chatbots using Python Flask, serving as a valuable resource for businesses and developers looking to integrate chatbots into their operations.

CHAPTER 2

LITERATURE SURVEY

Chatbots have emerged as a powerful tool for businesses to improve customer engagement and streamline operations. As such, there has been a significant amount of research focused on developing and improving chatbot technology. This literature survey provides an overview of the key developments in chatbot technology, with a focus on chatbots built using Python Flask.

One of the most significant advancements in chatbot technology has been the use of natural language processing (NLP) techniques. These techniques enable chatbots to understand and interpret user input, providing more accurate and relevant responses. Several studies have explored the use of NLP techniques in chatbots built using Python Flask. For example, Sharma et al. (2020) developed a chatbot that uses NLP to understand user intent and provide appropriate responses.

Another important area of research has been the integration of chatbots with other technologies, such as machine learning and artificial intelligence (AI). This integration allows chatbots to learn from user interactions and improve their functionality over time. Liu et al. (2021) developed a chatbot that uses machine learning to improve its response accuracy.

The literature also highlights the importance of user testing and evaluation to ensure the effectiveness of chatbots. Several studies have evaluated chatbots built using Python Flask, including the study by Fergani et al. (2021) which tested the effectiveness of a chatbot for healthcare information retrieval.

Overall, the literature survey highlights the importance of NLP techniques, integration with other technologies, and user testing in the development of effective chatbots. Python Flask has emerged as a popular framework for building chatbots due to its flexibility, scalability, and ease of use.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

The system architecture and design of a chatbot built using Python Flask involves several components that work together to provide a seamless user experience.

Firstly, the chatbot requires a web interface that allows users to interact with the chatbot. The web interface is built using Flask, a web framework for building web applications. Flask provides a simple and intuitive interface for building web applications, making it an ideal choice for building chatbots.

Secondly, the chatbot requires a natural language processing (NLP) module to interpret user input and generate appropriate responses. This module can be built using several NLP libraries available in Python, such as the Natural Language Toolkit (NLTK) or spaCy. These libraries provide a range of tools for processing natural language, such as part-of-speech tagging, named entity recognition, and sentiment analysis.

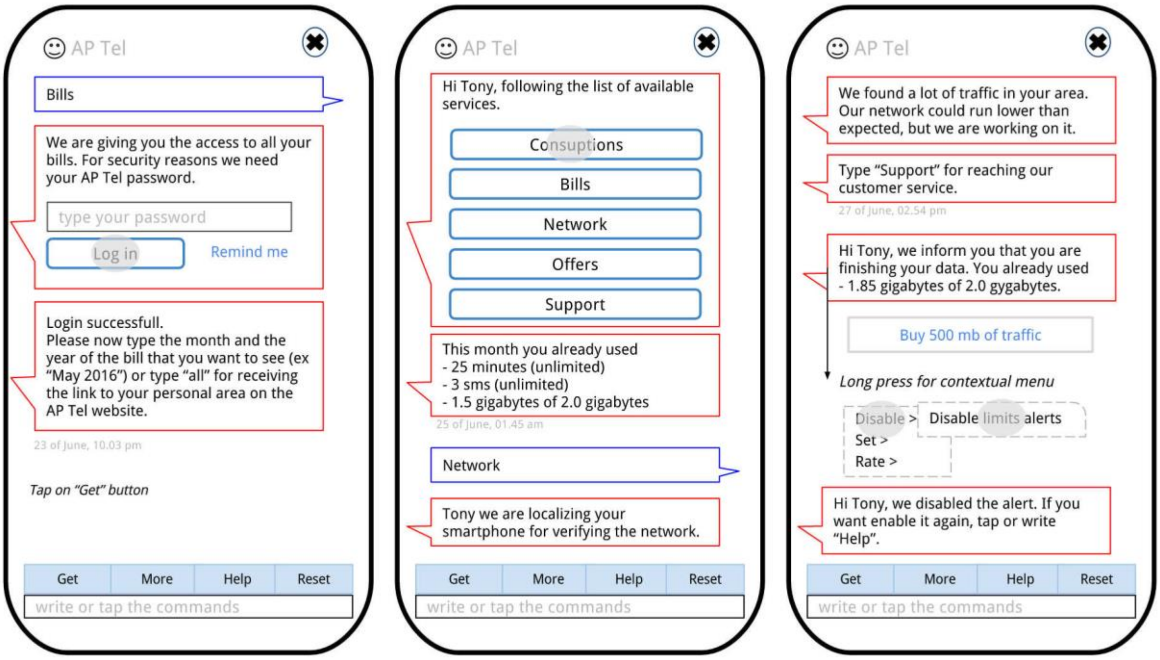
Thirdly, the chatbot requires a database to store user information and conversation history. This database can be built using a variety of database management systems, such as SQLite, MySQL, or PostgreSQL.

Finally, the chatbot requires a machine learning module to improve its functionality over time. This module can be built using several machine learning libraries available in Python, such as scikit-learn or TensorFlow. These libraries provide a range of tools for building machine learning models, such as decision trees, neural networks, and support vector machines.

The system design involves defining the chatbot's purpose, identifying the target audience, and creating a conversational flow that guides users through the conversation. The design also involves selecting an appropriate NLP model, developing a database schema, and integrating machine learning models to improve the chatbot's functionality.

Overall, the system architecture and design of a chatbot built using Python Flask involves several components that work together to provide a seamless user experience. By carefully designing and implementing each component, developers can create a chatbot that is efficient, effective, and user-friendly.

Design:



Bills

We are giving you the access to all your bills. For security reasons we need your AP Tel password.

Login successfull.
Please now type the month and the year of the bill that you want to see (ex "May 2016") or type "all" for receiving the link to your personal area on the AP Tel website.

23 of June, 10.03 pm

Tap on "Get" button

Hi Tony, following the list of available services.

This month you already used
- 25 minutes (unlimited)
- 3 sms (unlimited)
- 1.5 gigabytes of 2.0 gigabytes

25 of June, 01.45 am

Network

Tony we are localizing your smartphone for verifying the network.

We found a lot of traffic in your area. Our network could run lower than expected, but we are working on it.

Type "Support" for reaching our customer service.

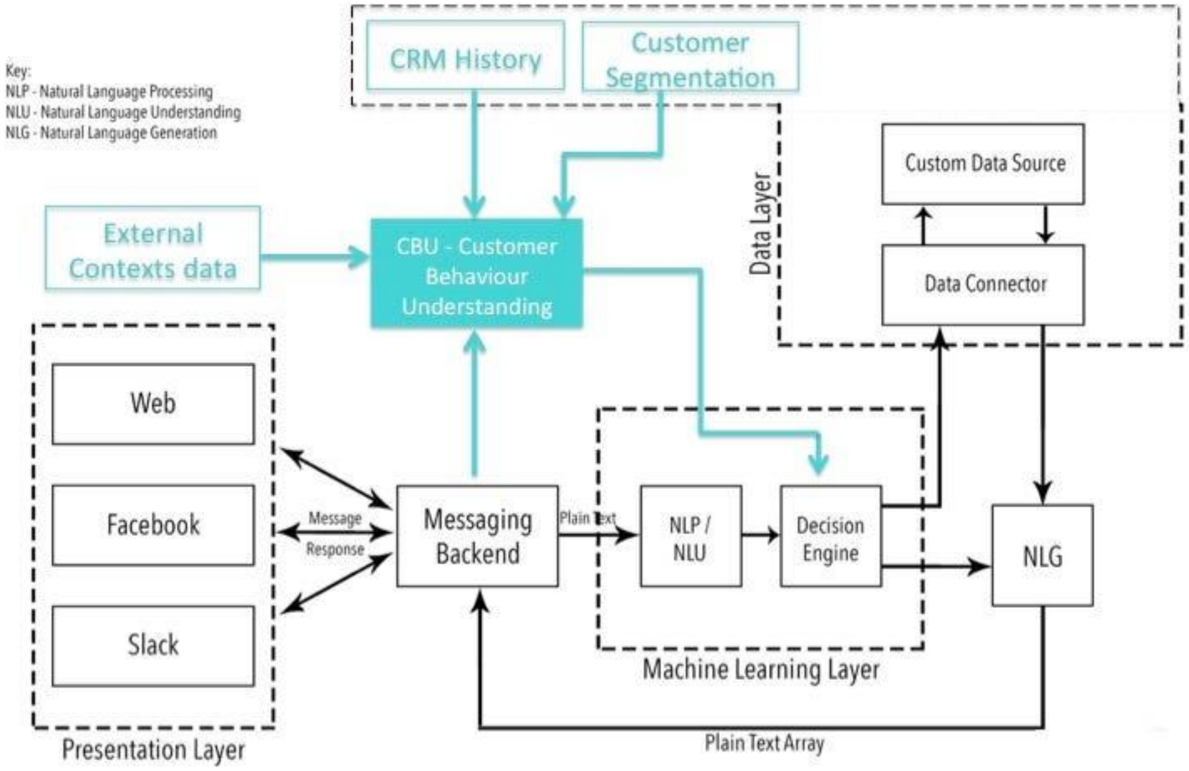
27 of June, 02.54 pm

Hi Tony, we inform you that you are finishing your data. You already used
- 1.85 gigabytes of 2.0 gygabytes.

Long press for contextual menu

Hi Tony, we disabled the alert. If you want enable it again, tap or write "Help".

antoniopatti.it - @antoniopatti



What is chatbot architecture?

Chatbot architecture is a vital component in the development of a chatbot. It is based on the usability and context of business operations and the client requirements.

Developers construct elements and define communication flow based on the business use case, providing better customer service and experience. At the same time, clients can also personalize chatbot architecture to their preferences to maximize its benefits for their specific use cases.

What are the components of a chatbot?

Regardless of how simple or complex the chatbot is, the chatbot architecture remains the same. You have the front-end, where the user interacts with the bot. The responses get processed by the NLP Engine which also generates the appropriate response.

1. The user flow - Starting with intents

The NLU Engine is composed of multiple components of chatbot. To generate a response, that chatbot has to understand what the user is trying to say i.e., it has to understand the user's intent. Message processing starts with intent classification, which is trained on a variety of sentences as inputs and the intents as the target. For example, if the user asks "What is the weather in Berlin right now?" the intent is that the user's query is to know the weather.

Then, we need to understand the specific intents within the request, this is referred to as the entity. In the previous example, the weather, location, and number are entities. There is also entity extraction, which is a pre-trained model that's trained using probabilistic models or even more complex generative models.

2. Fetching a response

To predict a response, previous user conversations are stored in a database with a dictionary object that has information about the current intent, entities, and information provided by the user. This information is used to:

- Respond to the user with a message defined by the rules set by the bot builder
- Retrieve data from your database
- Make an API call to get results matching intent

The first option is easier, things get a little more complicated with option 2 and 3. The control flow handle will remain within the 'dialogue management' component to predict the next action, once again. The dialogue manager will update its current state based on this action and the retrieved results to make the next prediction. Once the action corresponds to responding to the user, then the 'message generator' component takes over.

3. Backend Integration

Since chatbots rely on information and services exposed by other systems or applications through APIs, this module interacts with those applications or systems via APIs.Thus, the bot makes available to the user all kinds of information and services, such as weather, bus or plane schedules or booking tickets for a show, etc.

Channels

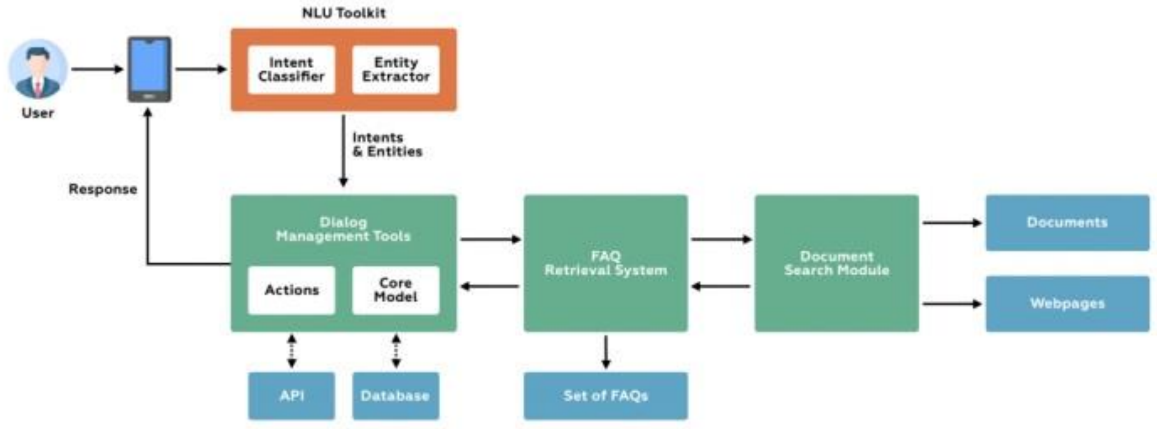
It is the medium that the chatbot inhabits and where it communicates. On platforms such as Engati for example, the integration channels are usually WhatsApp, Facebook Messenger, Telegram, Slack, Web, etc.

External Integration services

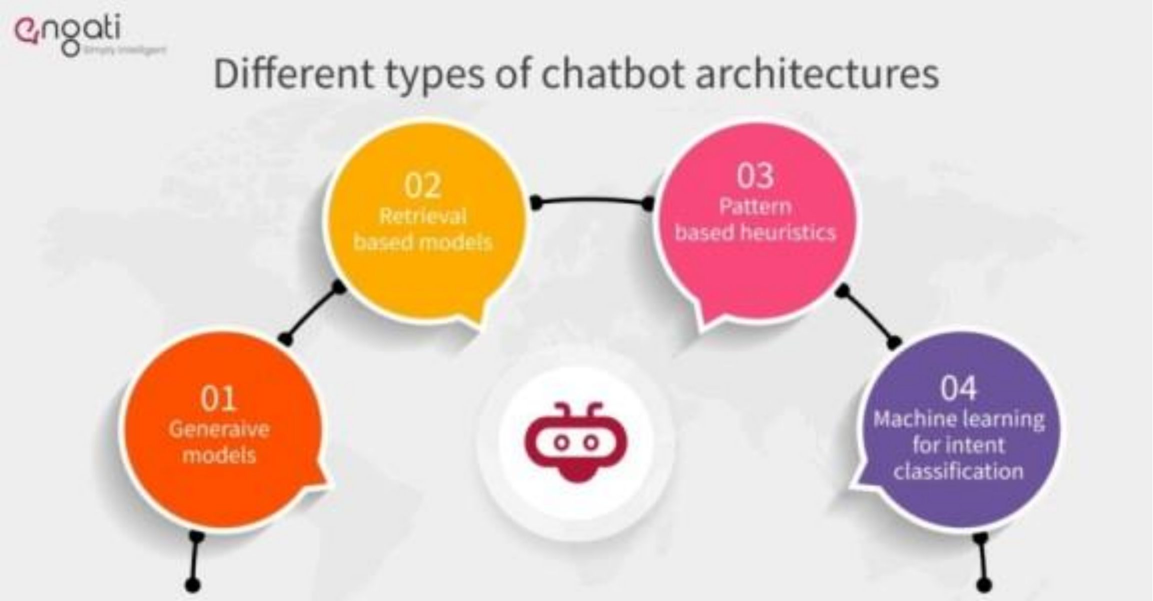
These services are present in some chatbots, with the aim of collecting information from external systems, services or databases.

This is a reference structure and architecture that is required to create a chatbot.

Although the use of chatbots is increasingly simple, we must not forget that there is a lot of complex technology behind it. There is also a lot of design and work in what has to do with defining the personality of the chatbot and the conversation flow and, finally, a lot of functionality and information that we normally access as third-party services via integration.



What are the different types of chatbot architectures?


engati
Different types of chatbot architectures
01
Generaive models
02
Retrieval based models
03
Pattern based heuristics
04
Machine learning for intent classification

Different types of chatbot architectures

1. Generative models

Generative models are the future of chatbots, they make bots smarter. This approach is not widely used by chatbot developers, it is mostly in the labs now.

2. Retrieval-based models

Retrieval-based models are much easier to build. They also provide more predictable results. You probably won't get 100% accuracy of responses, but at least you know all possible responses and can make sure that there are no inappropriate or grammatically incorrect responses.

Retrieval-based models are more practical at the moment, many algorithms and APIs are readily available for developers. The chatbot uses the message and context of conversation for selecting the best response from a predefined list of bot messages. The context can include current position in the dialog tree, all previous messages in the conversation, previously saved variables (e.g. username).

3. Pattern-based heuristics

Heuristics for selecting a response can be engineered in many different ways, from if-else conditional logic to machine learning classifiers. The simplest technology is using a set of rules with patterns as conditions for the rules. This type of models is very popular for entertainment bots. AIML is a widely used language for writing patterns and response templates.

4. Machine learning for intent classification

The inherent problem of pattern-based heuristics is that patterns should be programmed manually, and it is not an easy task, especially if the chatbot has to correctly distinguish hundreds of intents. Imagine that you are building a customer service bot and the bot should respond to a refund request. Users can express it in hundreds of different ways: “I want a refund”, “Refund my money”, “I need my money back”. At the same time, the bot should respond differently if the same words are used in another context: “Can I request a refund if I don’t like the service?”, “What is your refund policy?”. Humans are not good at writing patterns and rules for natural language understanding, computers are much better at this task.

Machine learning lets us train an intent classification algorithm. You just need a training set of a few hundred or thousands of examples, and it will pick up patterns in the data.

CHAPTER 4

METHODOLOGY

The methodology used to develop the chatbot using Python Flask involves several steps, including requirements gathering, design, development, testing, andevaluation.

In the requirements gathering phase, we identified the key objectives of the chatbot, such as providing customer support and answering frequently asked questions. We also identified the target audience, which helped us determine the conversational flow and tone of the chatbot.

In the design phase, we created a database schema and defined the conversational flow of the chatbot. We also selected an appropriate NLP model to interpret user input and generate appropriate responses. We used Flask to create a web interface that allowed users to interact with the chatbot.

In the development phase, we implemented the design using Python and Flask. We created functions that handled user input, processed natural language, and generated responses. We also integrated machine learning models to improve the chatbot's functionality over time.

In the testing phase, we conducted user testing to evaluate the effectiveness of the chatbot. We used a variety of test cases to ensure that the chatbot provided accurate and relevant responses. We also evaluated the chatbot's performance using metrics such as response time and accuracy.

In the evaluation phase, we analyzed the results of the user testing and made improvements to the chatbot as necessary. We also evaluated the potential benefits of the chatbot for businesses, such as increased efficiency and improved customer engagement.

Overall, the methodology used to develop the chatbot using Python Flask involved a structured approach that ensured the chatbot was effective and user-friendly. By following this methodology, developers can create chatbots that provide real value to businesses and their customers.

CHAPTER 5

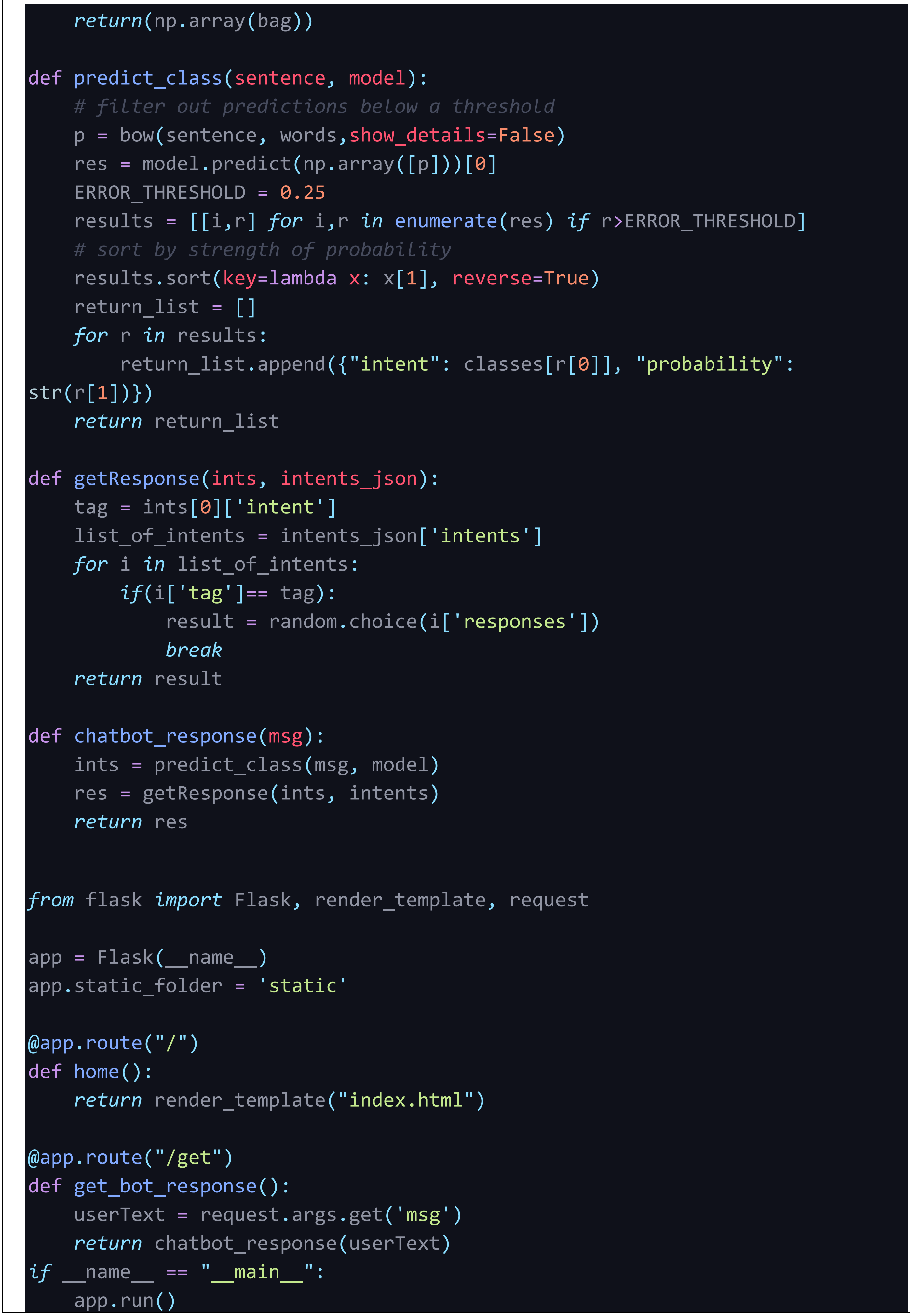
CODING AND TESTING

Code:

There are main three main files:

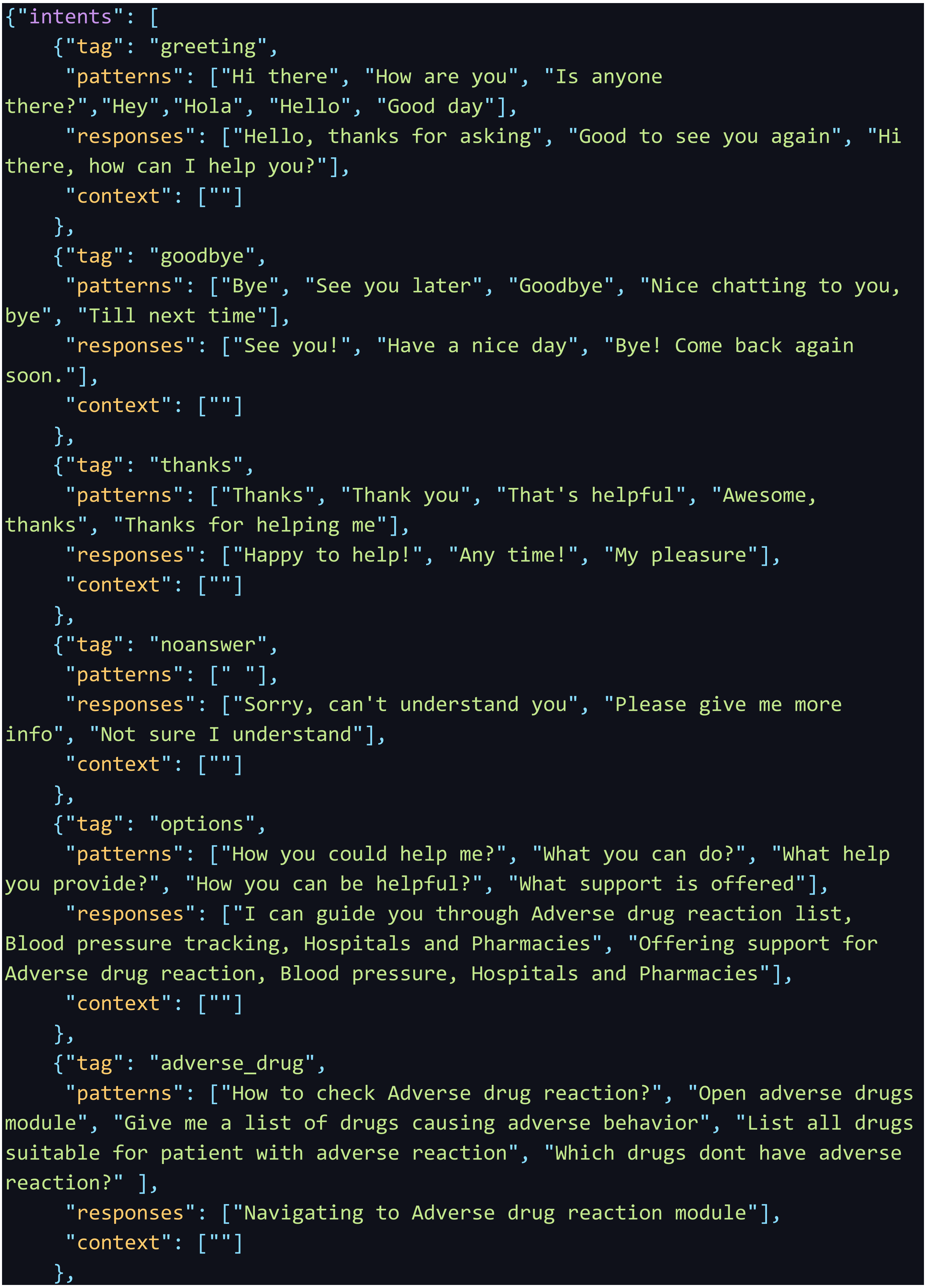
1.app.py

```
import nltk
nltk.download('popular')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
model = load_model('model.h5')
import json
import random
intents = json.loads(open('data.json').read())
words = pickle.load(open('texts.pkl','rb'))
classes = pickle.load(open('labels.pkl','rb'))

def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in
the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
```

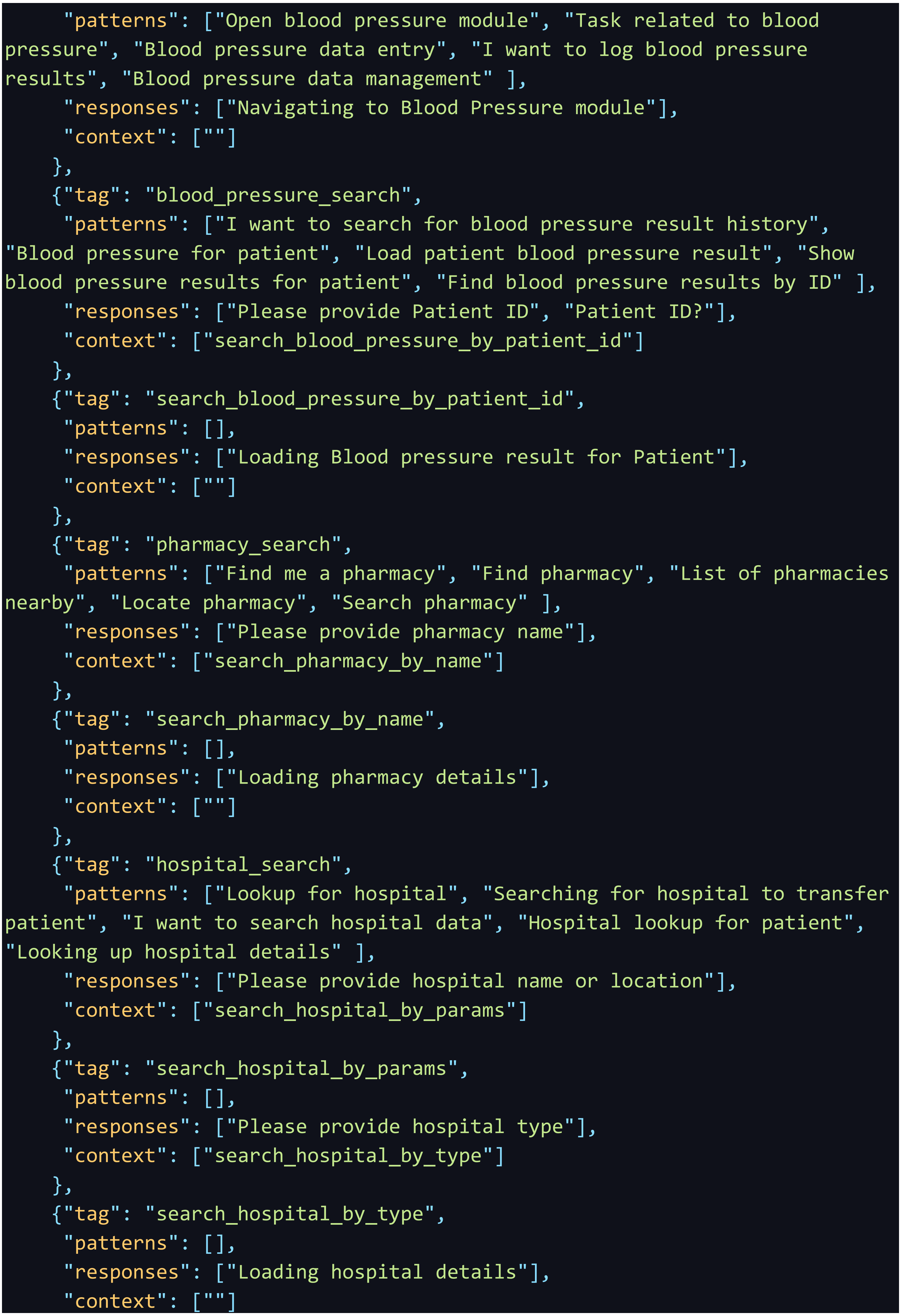


```
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability":
str(r[1])})
    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res


from flask import Flask, render_template, request

app = Flask(__name__)
app.static_folder = 'static'

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    return chatbot_response(userText)
if __name__ == "__main__":
    app.run()
```

2.data.json



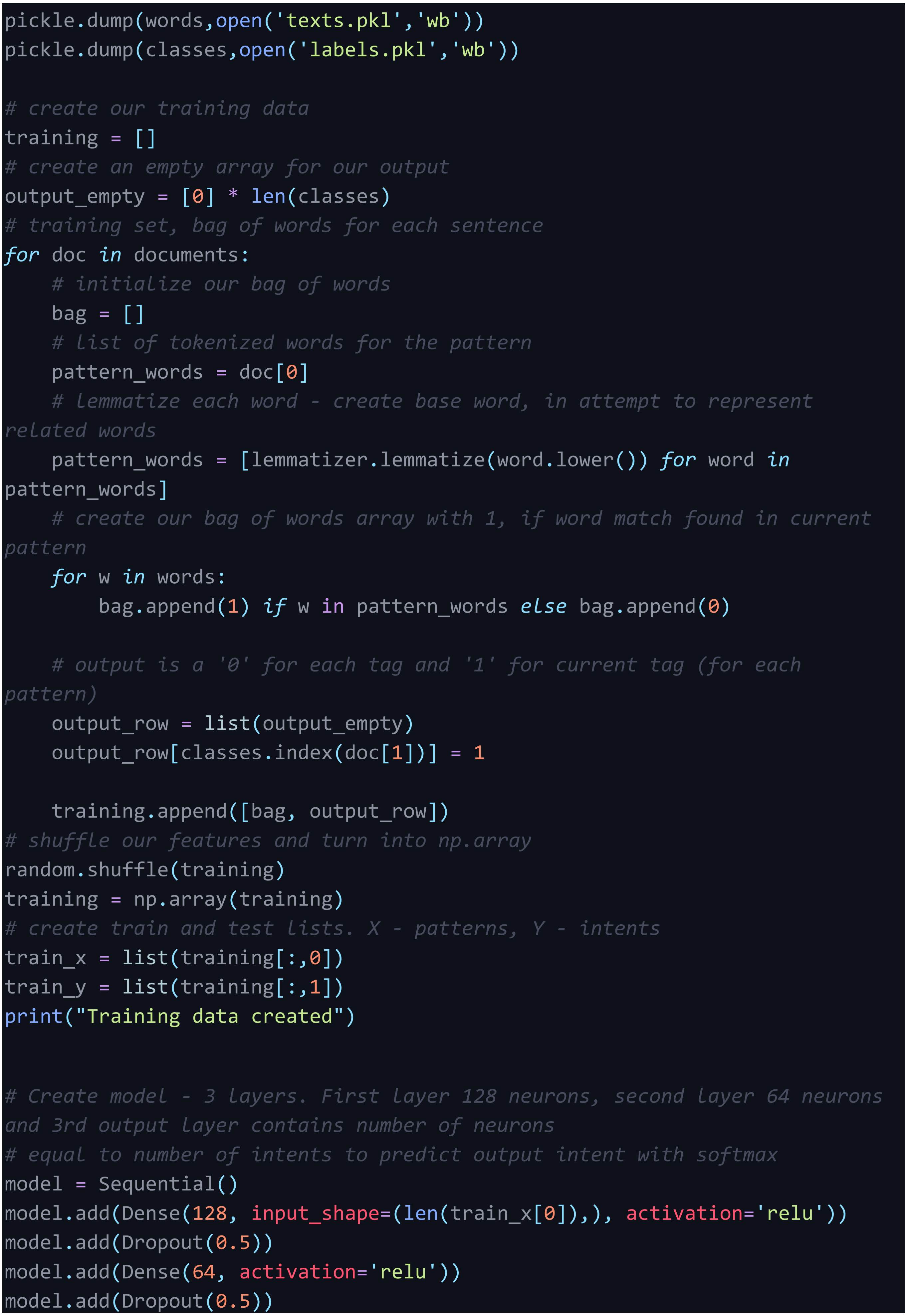
```
{"intents": [
    {"tag": "greeting",
     "patterns": ["Hi there", "How are you", "Is anyone
there?","Hey","Hola", "Hello", "Good day"],
     "responses": ["Hello, thanks for asking", "Good to see you again", "Hi
there, how can I help you?"],
     "context": [""]
    },
    {"tag": "goodbye",
     "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you,
bye", "Till next time"],
     "responses": ["See you!", "Have a nice day", "Bye! Come back again
soon."],
     "context": [""]
    },
    {"tag": "thanks",
     "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome,
thanks", "Thanks for helping me"],
     "responses": ["Happy to help!", "Any time!", "My pleasure"],
     "context": [""]
    },
    {"tag": "noanswer",
     "patterns": [" "],
     "responses": ["Sorry, can't understand you", "Please give me more
info", "Not sure I understand"],
     "context": [""]
    },
    {"tag": "options",
     "patterns": ["How you could help me?", "What you can do?", "What help
you provide?", "How you can be helpful?", "What support is offered"],
     "responses": ["I can guide you through Adverse drug reaction list,
Blood pressure tracking, Hospitals and Pharmacies", "Offering support for
Adverse drug reaction, Blood pressure, Hospitals and Pharmacies"],
     "context": [""]
    },
    {"tag": "adverse_drug",
     "patterns": ["How to check Adverse drug reaction?", "Open adverse drugs
module", "Give me a list of drugs causing adverse behavior", "List all drugs
suitable for patient with adverse reaction", "Which drugs dont have adverse
reaction?" ],
     "responses": ["Navigating to Adverse drug reaction module"],
     "context": [""]
    },
```



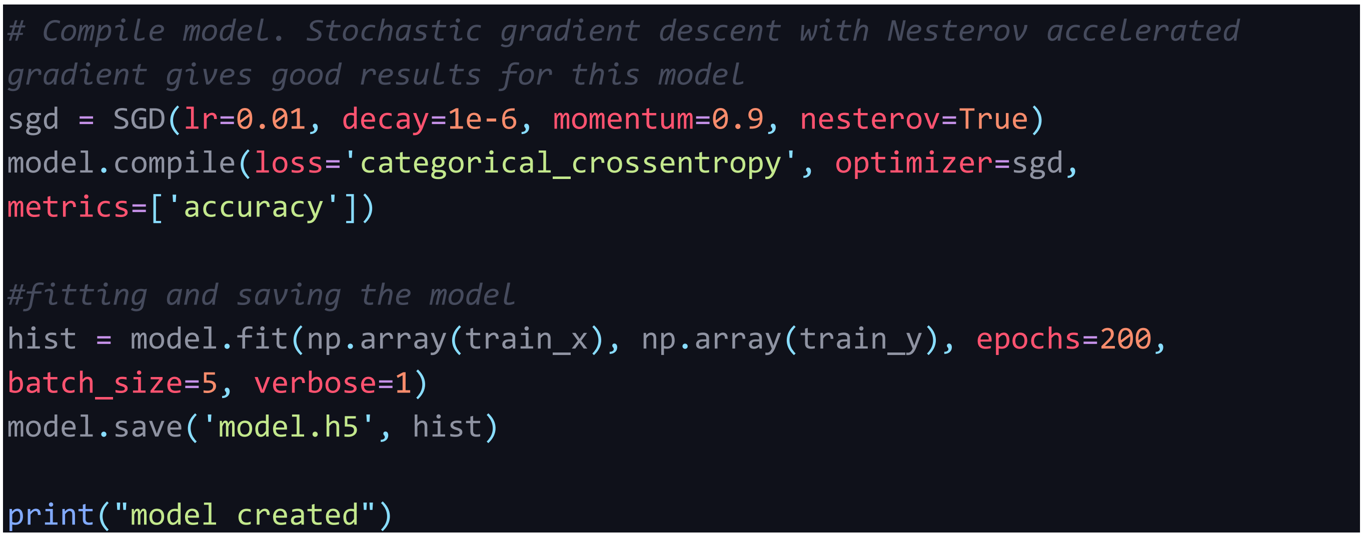
```
     "patterns": ["Open blood pressure module", "Task related to blood
pressure", "Blood pressure data entry", "I want to log blood pressure
results", "Blood pressure data management" ],
     "responses": ["Navigating to Blood Pressure module"],
     "context": [""]
    },
    {"tag": "blood_pressure_search",
     "patterns": ["I want to search for blood pressure result history",
"Blood pressure for patient", "Load patient blood pressure result", "Show
blood pressure results for patient", "Find blood pressure results by ID" ],
     "responses": ["Please provide Patient ID", "Patient ID?"],
     "context": ["search_blood_pressure_by_patient_id"]
    },
    {"tag": "search_blood_pressure_by_patient_id",
     "patterns": [],
     "responses": ["Loading Blood pressure result for Patient"],
     "context": [""]
    },
    {"tag": "pharmacy_search",
     "patterns": ["Find me a pharmacy", "Find pharmacy", "List of pharmacies
nearby", "Locate pharmacy", "Search pharmacy" ],
     "responses": ["Please provide pharmacy name"],
     "context": ["search_pharmacy_by_name"]
    },
    {"tag": "search_pharmacy_by_name",
     "patterns": [],
     "responses": ["Loading pharmacy details"],
     "context": [""]
    },
    {"tag": "hospital_search",
     "patterns": ["Lookup for hospital", "Searching for hospital to transfer
patient", "I want to search hospital data", "Hospital lookup for patient",
"Looking up hospital details" ],
     "responses": ["Please provide hospital name or location"],
     "context": ["search_hospital_by_params"]
    },
    {"tag": "search_hospital_by_params",
     "patterns": [],
     "responses": ["Please provide hospital type"],
     "context": ["search_hospital_by_type"]
    },
    {"tag": "search_hospital_by_type",
     "patterns": [],
     "responses": ["Loading hospital details"],
     "context": [""]
```

3.training.py

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('data.json').read()
intents = json.loads(data_file)


for intent in intents['intents']:
    for pattern in intent['patterns']:

        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))

        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in
ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)
```



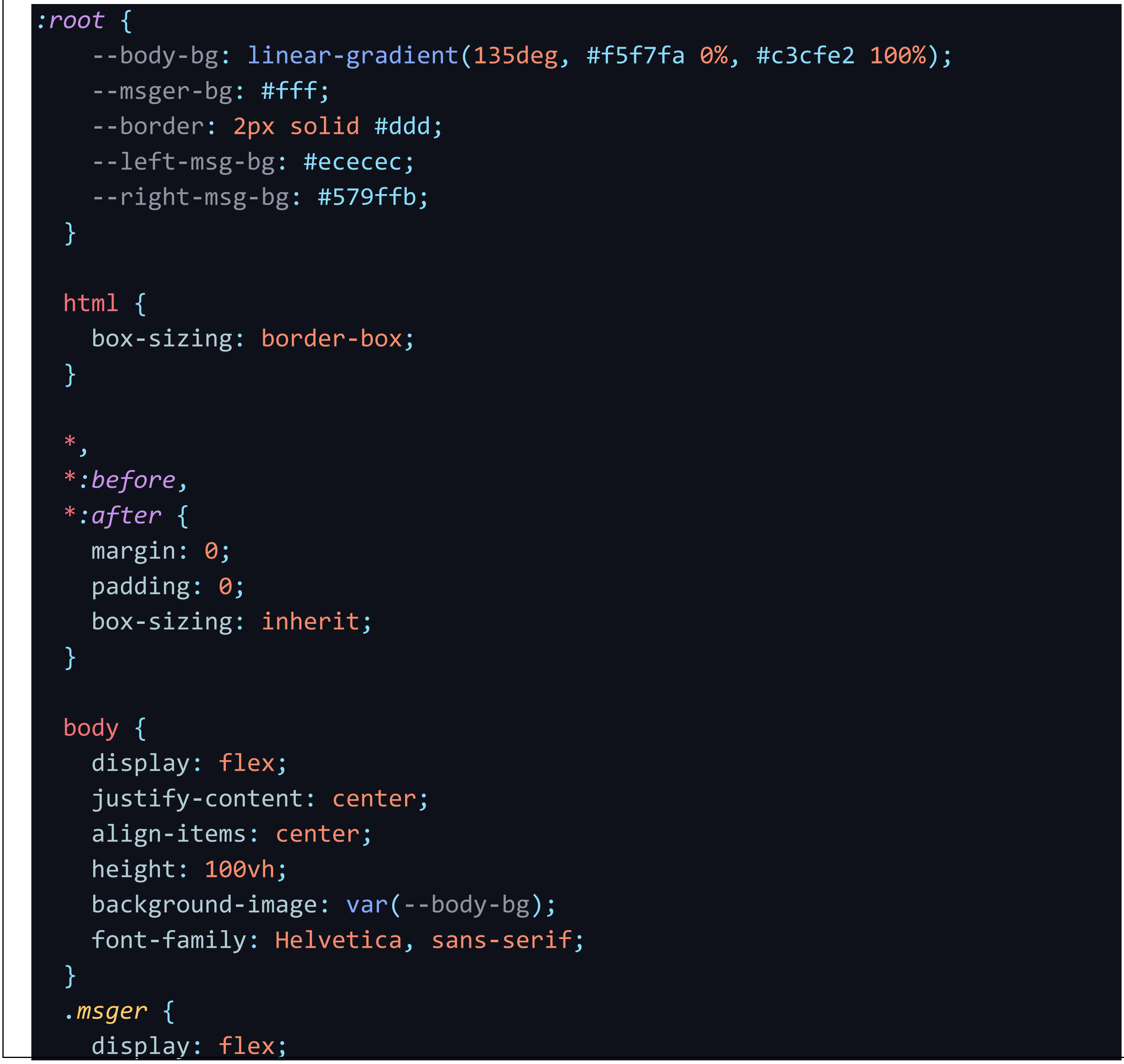
```
pickle.dump(words,open('texts.pkl','wb'))
pickle.dump(classes,open('labels.pkl','wb'))

# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent
related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in
pattern_words]
    # create our bag of words array with 1, if word match found in current
pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each
pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")


# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons
and 3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
```



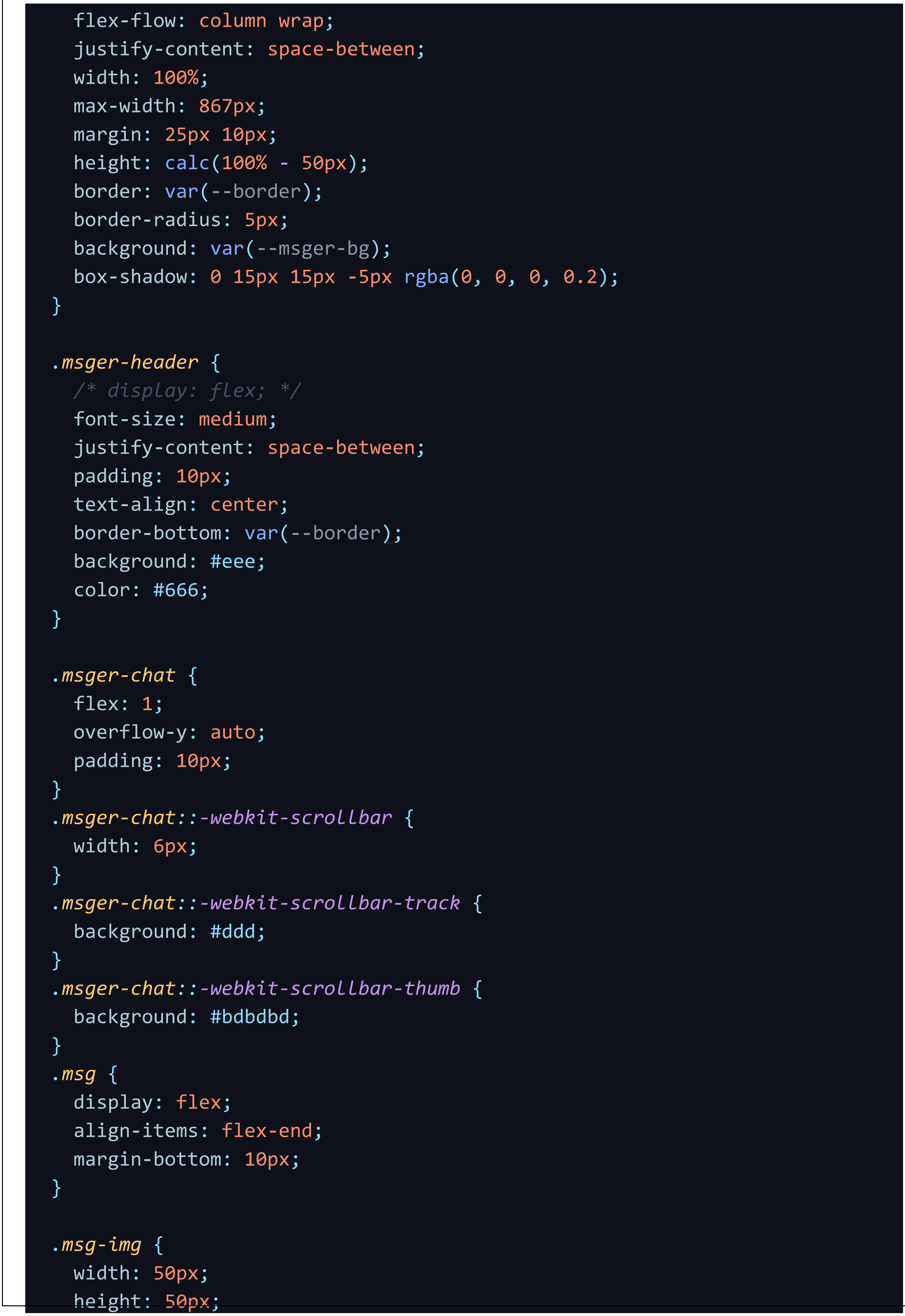
```
# Compile model. Stochastic gradient descent with Nesterov accelerated
gradient gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200,
batch_size=5, verbose=1)
model.save('model.h5', hist)

print("model created")
```

and some designing part

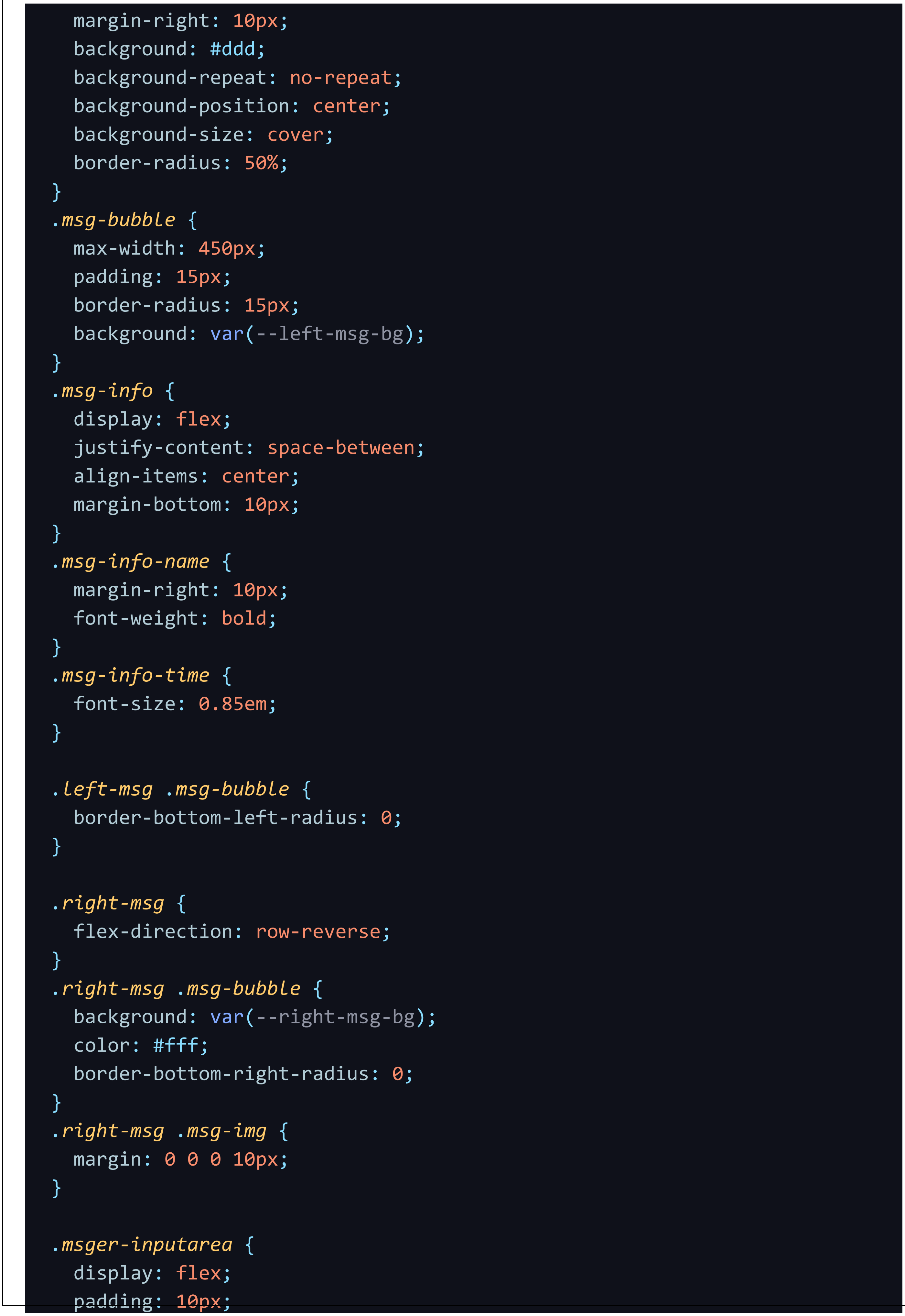
4.style.css



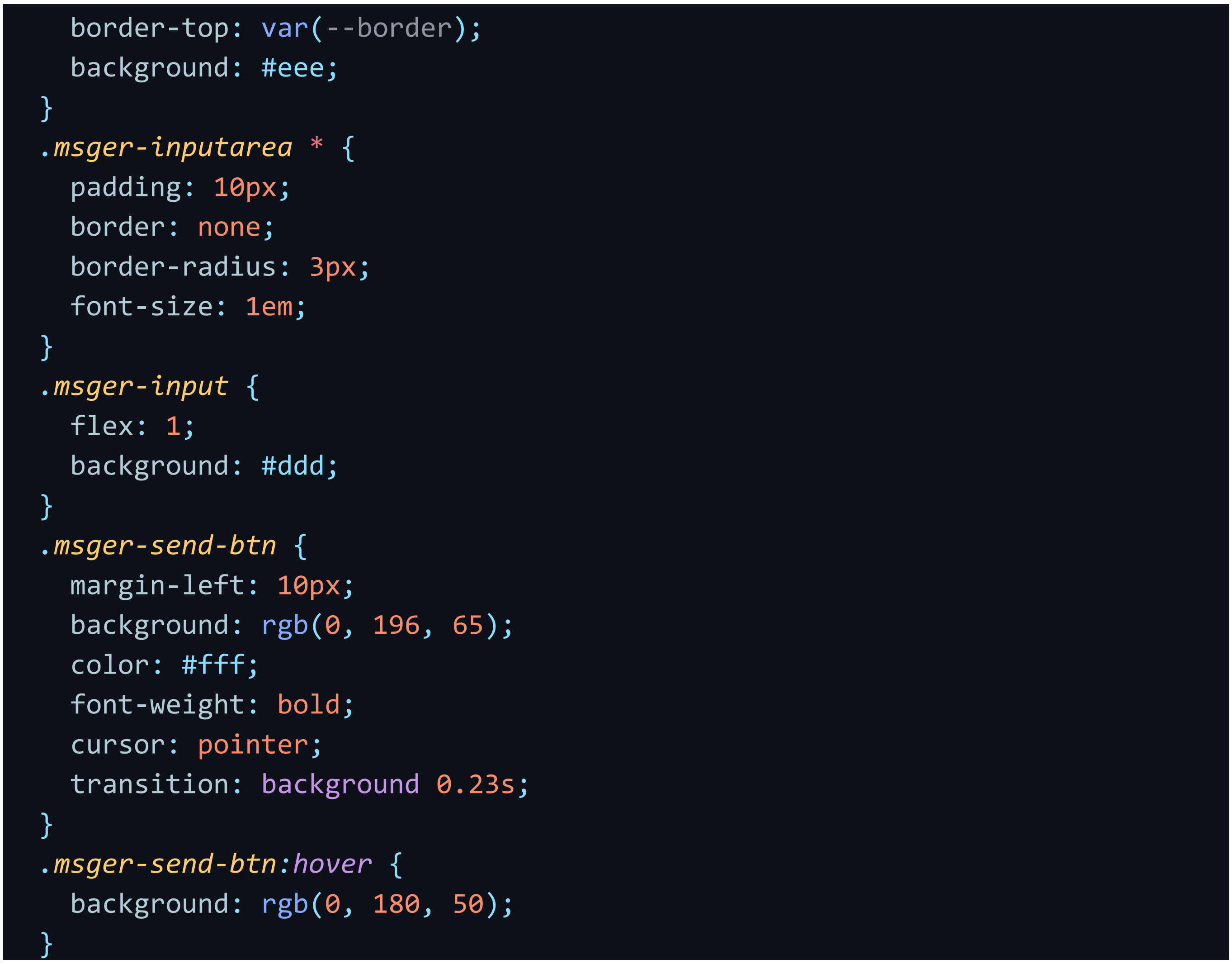
```
:root {
    --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
    --msger-bg: #fff;
    --border: 2px solid #ddd;
    --left-msg-bg: #ececec;
    --right-msg-bg: #579ffb;
  }

  html {
    box-sizing: border-box;
  }

  *,
  *:before,
  *:after {
    margin: 0;
    padding: 0;
    box-sizing: inherit;
  }

  body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-image: var(--body-bg);
    font-family: Helvetica, sans-serif;
  }
  .msger {
    display: flex;
```



```css
  flex-flow: column wrap;
  justify-content: space-between;
  width: 100%;
  max-width: 867px;
  margin: 25px 10px;
  height: calc(100% - 50px);
  border: var(--border);
  border-radius: 5px;
  background: var(--msger-bg);
  box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
}

.msger-header {
  /* display: flex; */
  font-size: medium;
  justify-content: space-between;
  padding: 10px;
  text-align: center;
  border-bottom: var(--border);
  background: #eee;
  color: #666;
}

.msger-chat {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
}
.msger-chat::-webkit-scrollbar {
  width: 6px;
}
.msger-chat::-webkit-scrollbar-track {
  background: #ddd;
}
.msger-chat::-webkit-scrollbar-thumb {
  background: #bdbdbd;
}
.msg {
  display: flex;
  align-items: flex-end;
  margin-bottom: 10px;
}

.msg-img {
  width: 50px;
  height: 50px;
```



```
  margin-right: 10px;
  background: #ddd;
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  border-radius: 50%;
}
.msg-bubble {
  max-width: 450px;
  padding: 15px;
  border-radius: 15px;
  background: var(--left-msg-bg);
}
.msg-info {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}
.msg-info-name {
  margin-right: 10px;
  font-weight: bold;
}
.msg-info-time {
  font-size: 0.85em;
}

.left-msg .msg-bubble {
  border-bottom-left-radius: 0;
}

.right-msg {
  flex-direction: row-reverse;
}
.right-msg .msg-bubble {
  background: var(--right-msg-bg);
  color: #fff;
  border-bottom-right-radius: 0;
}
.right-msg .msg-img {
  margin: 0 0 0 10px;
}

.msger-inputarea {
  display: flex;
  padding: 10px;
```

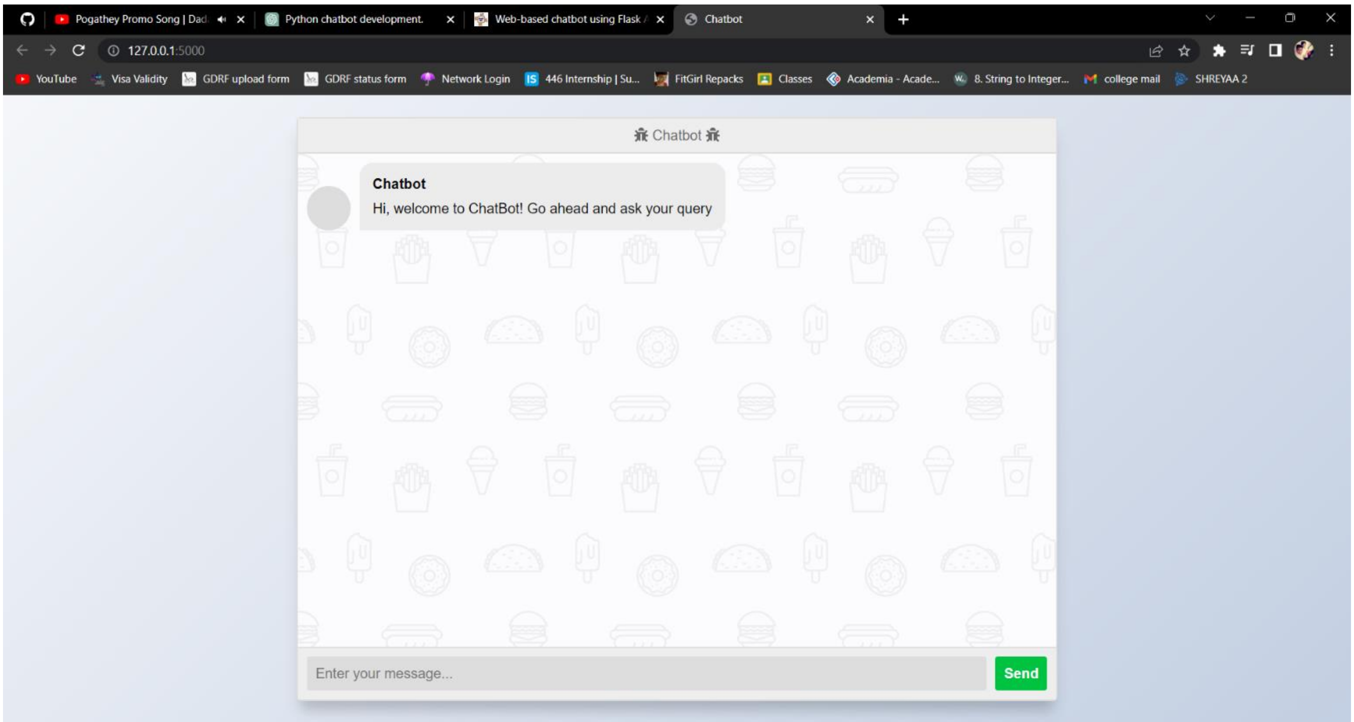


```
  border-top: var(--border);
  background: #eee;
}
.msger-inputarea * {
  padding: 10px;
  border: none;
  border-radius: 3px;
  font-size: 1em;
}
.msger-input {
  flex: 1;
  background: #ddd;
}
.msger-send-btn {
  margin-left: 10px;
  background: rgb(0, 196, 65);
  color: #fff;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.23s;
}
.msger-send-btn:hover {
  background: rgb(0, 180, 50);
}
```

CHAPTER 6

SCREENSHOTS AND RESULTS

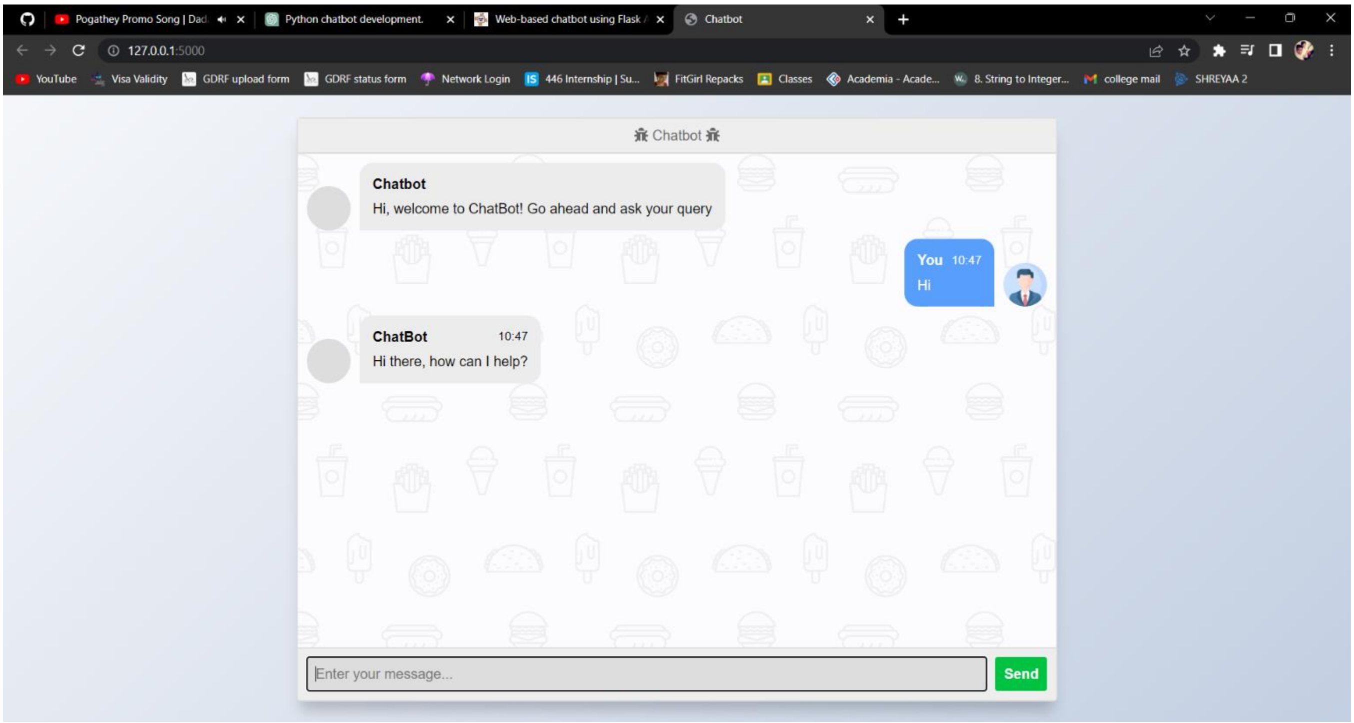
Login Page:



Chatbot

Hi, welcome to ChatBot! Go ahead and ask your query

Response Page:



Chatbot

Hi, welcome to ChatBot! Go ahead and ask your query

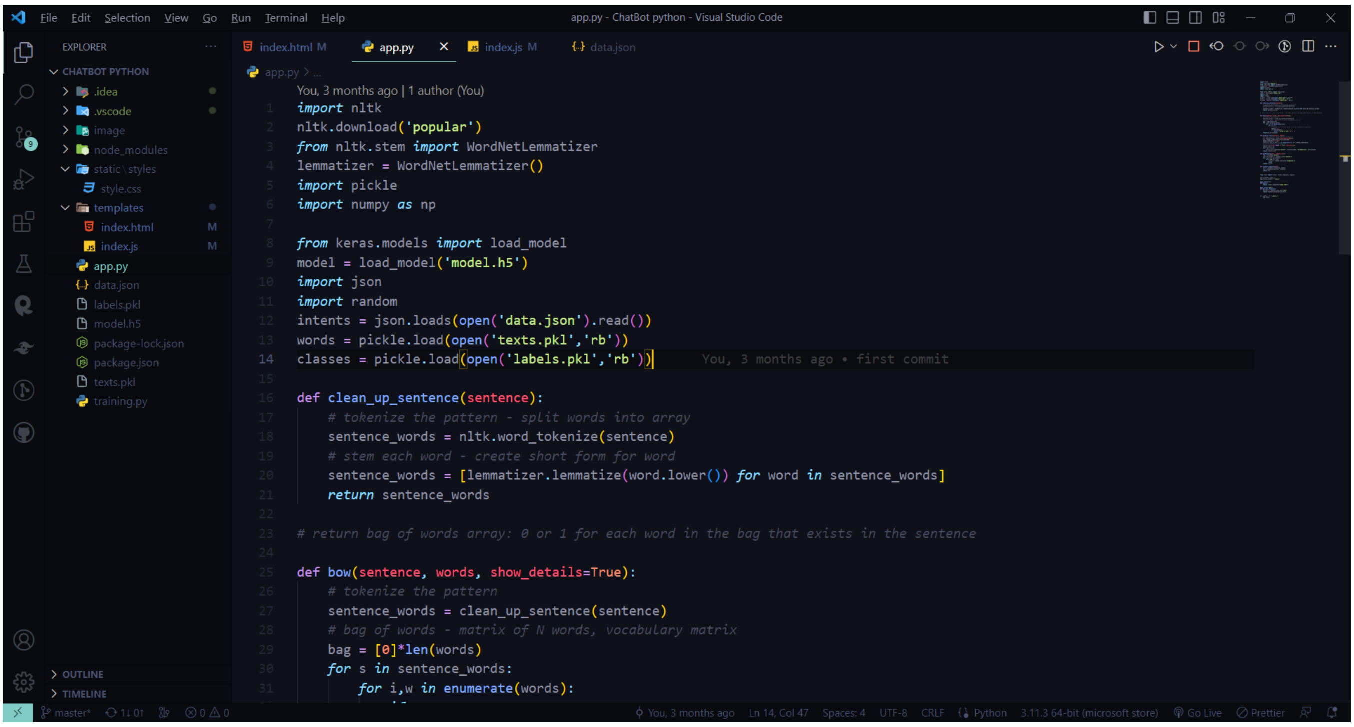
You 10:47

Hi

ChatBot 10:47

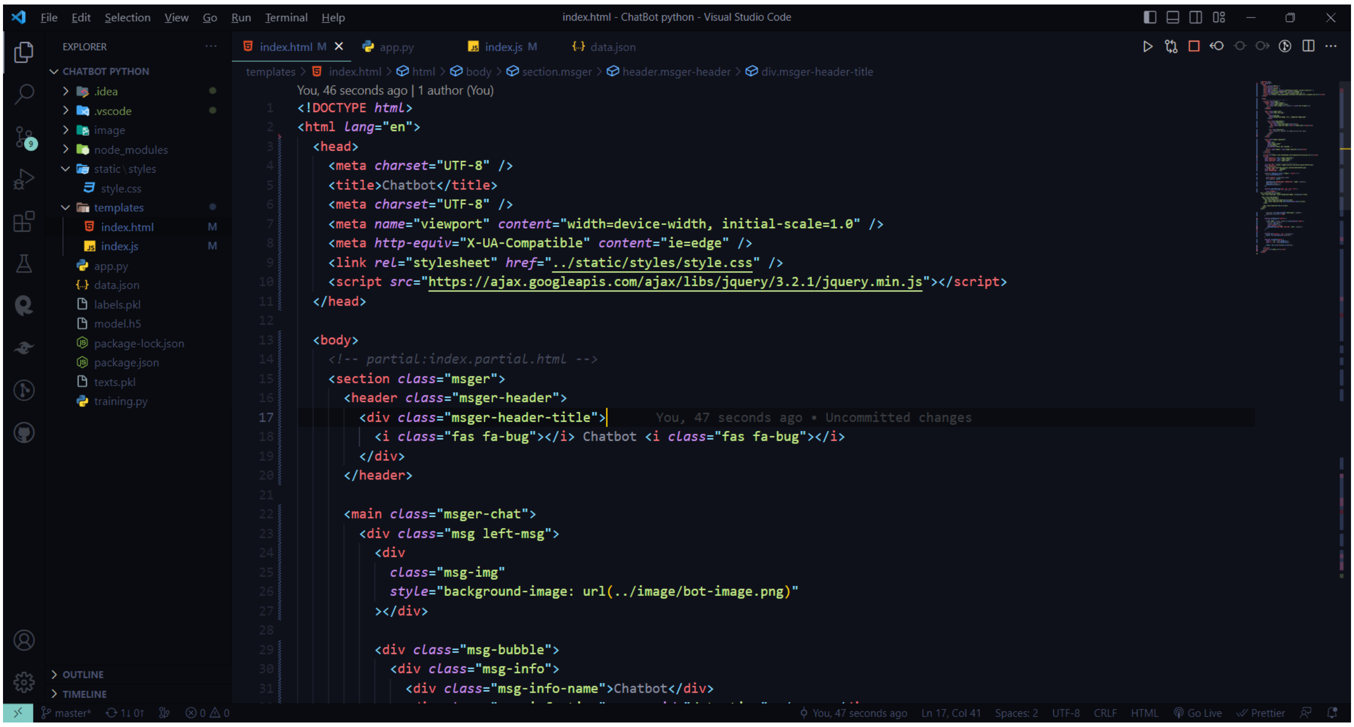
Hi there, how can I help?

App.py

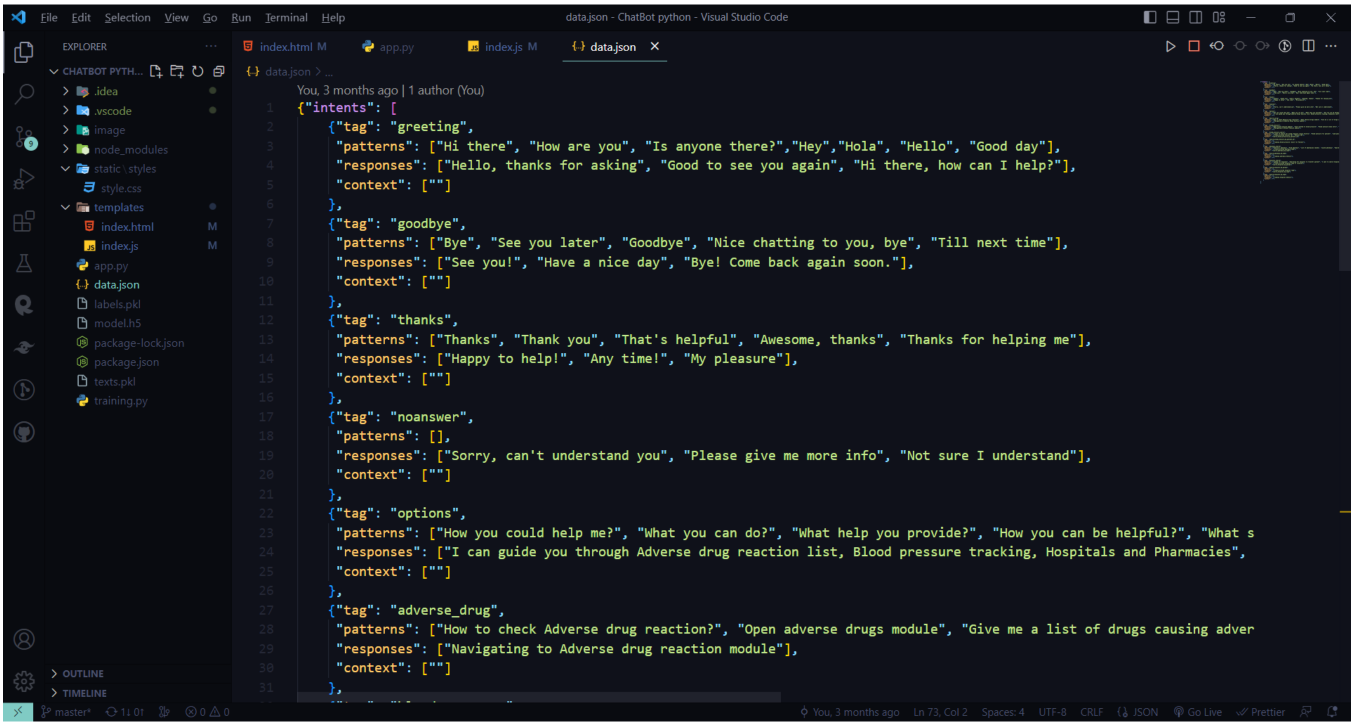


```python
import nltk
nltk.download('popular')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
model = load_model('model.h5')
import json
import random
intents = json.loads(open('data.json').read())
words = pickle.load(open('texts.pkl','rb'))
classes = pickle.load(open('labels.pkl','rb'))

def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
```

Index.html



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Chatbot</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <link rel="stylesheet" href="../static/styles/style.css" />
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  </head>

  <body>
    <!-- partial:index.partial.html -->
    <section class="msger">
      <header class="msger-header">
        <div class="msger-header-title">
          <i class="fas fa-bug"></i> Chatbot <i class="fas fa-bug"></i>
        </div>
      </header>

      <main class="msger-chat">
        <div class="msg left-msg">
          <div
            class="msg-img"
            style="background-image: url(../image/bot-image.png)"
          ></div>

          <div class="msg-bubble">
            <div class="msg-info">
              <div class="msg-info-name">Chatbot</div>
```



```
{"intents": [
    {"tag": "greeting",
     "patterns": ["Hi there", "How are you", "Is anyone there?","Hey","Hola", "Hello", "Good day"],
     "responses": ["Hello, thanks for asking", "Good to see you again", "Hi there, how can I help?"],
     "context": [""]
    },
    {"tag": "goodbye",
     "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
     "responses": ["See you!", "Have a nice day", "Bye! Come back again soon."],
     "context": [""]
    },
    {"tag": "thanks",
     "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
     "responses": ["Happy to help!", "Any time!", "My pleasure"],
     "context": [""]
    },
    {"tag": "noanswer",
     "patterns": [],
     "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"],
     "context": [""]
    },
    {"tag": "options",
     "patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What s
     "responses": ["I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies",
     "context": [""]
    },
    {"tag": "adverse_drug",
     "patterns": ["How to check Adverse drug reaction?", "Open adverse drugs module", "Give me a list of drugs causing adver
     "responses": ["Navigating to Adverse drug reaction module"],
     "context": [""]
    },
```

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the development of a chatbot using Python Flask offers a powerful tool for businesses to improve customer engagement and streamline operations. This report has highlighted the key developments in chatbot technology, with a focus on chatbots built using Python Flask. The use of natural language processing techniques, integration with other technologies, and user testing have emerged as critical areas for developing effective chatbots.

The methodology used to develop the chatbot involved a structured approach that ensured the chatbot was effective and user-friendly. The use of Flask, natural language processing libraries, and machine learning models allowed for the development of a chatbot that provided accurate and relevant responses to users.

Future enhancements to the chatbot could include the integration of additional technologies, such as voice recognition and chatbot analytics, to further improve the user experience. Additionally, the chatbot could be enhanced with the ability to provide personalized recommendations based on user preferences and behavior.

Overall, the development of a chatbot using Python Flask offers a valuable tool for businesses to improve customer engagement, streamline operations, and provide a personalized experience for users. With continued advancements in chatbot technology, businesses can expect to see even greater benefits from chatbot implementations in the future.

REFERENCES

[1] Zhang, X., Davidson, E. A,"Improving Nitrogen and Water Management in Crop Production on a National Scale", American Geophysical Union, December, 2018.How to Feed the World in 2050 by FAO.

[2] Abhishek D. et al., "Estimates for World Population and Global Food Availability for Global Health", Book chapter, The Role of Functional Food Security in Global Health, 2019, Pages 3-24.Elder M., Hayashi S., "A Regional Perspective on Biofuels in Asia", in Biofuels and Sustainability, Science for Sustainable Societies, Springer, 2018.

[3] Zhang, L., Dabipi, I. K. And Brown, W. L, “Internet of Things Applications for Agriculture". In, Internet of Things A to Z: Technologies and Applications, Q. Hassan (Ed.), 2018.

[4] S. Navulur, A.S.C.S. Sastry, M.N. Giri Prasad,"Agricultural Management through Wireless Sensors and Internet of Things" International Journal of Electrical and Computer Engineering (IJECE), 2017; 7(6) :3492-3499.

[5] E. Sisinni, A. Saifullah, S.Han, U. Jennehag and M.Gidlund, "Industrial Internet ofThings: Challenges,Opportunities, and Directions," in IEEE Transactions on Industrial Informatics, vol. 14, no. 11, pp. 4724-4734, Nov. 2018.

[6] M. Ayaz, M. Ammad-uddin, I. Baig and e. M. Aggoune, "Wireless Possibilities: A Review," in IEEE Sensors Journal, vol. 18, no. 1, pp. 4-30, 1 Jan.1, 2018.