FAQ Chatbot with Rasa

I'll create a complete FAQ chatbot using Python and Rasa that can handle predefined queries. This solution includes the necessary Rasa files and a simple interface to interact with the bot.

Project Structure

```
faq-chatbot/
├── data/
│   ├── nlu.yml
│   ├── rules.yml
│   └── stories.yml
├── config.yml
├── credentials.yml
├── domain.yml
├── endpoints.yml
├── app.py
└── requirements.txt
```

Implementation

Let me create the necessary files for this project:

1. requirements.txt

```txt
rasa==3.6.15
spacy==3.7.2
ujson==5.8.0
python-socketio==5.10.0
```

2. data/nlu.yml

```yaml
version: "3.1"

nlu:
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    - hello there
    - good morning
    - good evening
    - moin
    - hey there
    - let's go
    - hey dude
    - goodmorning
    - goodevening
    - good afternoon

- intent: goodbye
  examples: |
    - cu
    - good by
    - cee you later
    - good night
    - bye
    - goodbye
    - have a nice day
    - see you around
    - bye bye
    - see you later

- intent: affirm
  examples: |
    - yes
    - y
    - indeed
    - of course
    - that sounds good
    - correct
- intent: deny
  examples: |
    - no
    - n
    - never
    - I don't think so
    - don't like that
    - no way
    - not really

- intent: faq_returns
  examples: |
    - What is your return policy?
    - How do I return an item?
    - Can I get a refund?
    - What if I want to return something?
    - Do you accept returns?
    - How long do I have to return an item?
    - Return policy

- intent: faq_shipping
  examples: |
    - How long does shipping take?
    - What are your shipping options?
    - When will my order arrive?
    - Do you ship internationally?
    - Shipping times
    - How much is shipping?
    - Delivery options

- intent: faq_products
  examples: |
    - What products do you offer?
    - What items do you have?
    - What do you sell?
    - Tell me about your products
    - Product catalog
    - What kind of products are available?

- intent: faq_payment
  examples: |
    - What payment methods do you accept?
    - How can I pay?
    - Do you take credit cards?
    - Payment options
    - Can I pay with PayPal?
    - What forms of payment do you accept?

- intent: faq_contact
  examples: |
    - How can I contact you?
    - What's your phone number?
    - Where are you located?
    - How do I reach customer service?
    - Contact information
    - Customer support
    - Help line
```

3. data/stories.yml

```yaml
version: "3.1"

stories:
- story: greet and faq
  steps:
  - intent: greet
  - action: utter_greet
  - intent: faq_returns
  - action: utter_returns

- story: returns policy
  steps:
  - intent: faq_returns
  - action: utter_returns
- story: shipping info
  steps:
  - intent: faq_shipping
  - action: utter_shipping

- story: product info
  steps:
  - intent: faq_products
  - action: utter_products

- story: payment info
  steps:
  - intent: faq_payment
  - action: utter_payment

- story: contact info
  steps:
  - intent: faq_contact
  - action: utter_contact

- story: goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

4. data/rules.yml

```yaml
version: "3.1"

rules:
- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye

- rule: Say 'I am a bot' anytime the user challenges
  steps:
  - intent: greet
  - action: utter_greet
```

5. config.yml

```yaml
recipe: default.v1
language: en
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
```

```
 - name: CountVectorsFeaturizer
 - name: CountVectorsFeaturizer
   analyzer: char_wb
   min_ngram: 1
   max_ngram: 4
 - name: DIETClassifier
   epochs: 100
   constrain_similarities: true
 - name: EntitySynonymMapper
 - name: ResponseSelector
   epochs: 100
   constrain_similarities: true
 - name: FallbackClassifier
   threshold: 0.3
   ambiguity_threshold: 0.1

policies:
 - name: MemoizationPolicy
 - name: RulePolicy
   core_fallback_threshold: 0.3
   core_fallback_action_name: action_default_fallback
   enable_fallback_prediction: true
 - name: UnexpecTEDIntentPolicy
   max_history: 5
   epochs: 100
 - name: TEDPolicy
```

```
   max_history: 5
   epochs: 100
   constrain_similarities: true
```

6. domain.yml

```yaml
version: "3.1"

intents:
  - greet
  - goodbye
  - affirm
  - deny
  - faq_returns
  - faq_shipping
  - faq_products
  - faq_payment
  - faq_contact

responses:
  utter_greet:
  - text: "Hello! How can I help you today?"

  utter_goodbye:
```

```
  - text: "Goodbye! Have a great day!"
  - text: "See you later!"
  - text: "Bye! Come back if you have more questions."

  utter_returns:
  - text: "Our return policy allows returns within 30 days of purchase. Items must be in
original condition with tags attached. Please contact our support team to initiate a return."

  utter_shipping:
  - text: "We offer standard shipping (3-5 business days) for $5.99 and express shipping (1-2
business days) for $12.99. International shipping is also available for an additional fee."

  utter_products:
  - text: "We offer a wide range of products including electronics, clothing, home goods, and
beauty products. Is there a specific category you're interested in?"

  utter_payment:
  - text: "We accept all major credit cards (Visa, MasterCard, American Express), PayPal,
and Apple Pay. All payments are processed securely."

  utter_contact:
  - text: "You can contact us at support@company.com or call us at 1-800-123-4567. Our
customer service hours are Monday-Friday, 9am-5pm EST."

  utter_default:
  - text: "I'm sorry, I didn't understand that. Could you please rephrase?"
```

```
session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

7. endpoints.yml

```yaml
# This file contains the different endpoints your bot can use.

# Server where the models are pulled from.
# https://rasa.com/docs/rasa/model-storage#fetching-models-from-a-server

#models:
#  url: http://my-server.com/models/default_core@latest
#  wait_time_between_pulls: 10   # [optional](default: 100)

# Server which runs your custom actions.
# https://rasa.com/docs/rasa/custom-actions

#action_endpoint:
#  url: "http://localhost:5055/webhook"

# Tracker store which is used to store the conversations.
# https://rasa.com/docs/rasa/tracker-stores
```

```
#tracker_store:
#    type: SQL
#    dialect: sqlite
#    db: storage.db

# Event broker which is used to stream events to other applications.
# https://rasa.com/docs/rasa/event-brokers

#event_broker:
#  url: localhost
#  username: username
#  password: password
#  queue: queue
```

8. credentials.yml

```yaml
# This file contains the credentials for the voice & chat platforms
# which your bot is using.
# https://rasa.com/docs/rasa/messaging-and-voice-channels

rest:
#  # you don't need to provide anything here - this channel doesn't
#  # require any credentials
```

```
#facebook:
#  verify: "<verify>"
#  secret: "<your secret>"
#  page-access-token: "<your page access token>"

#slack:
#  slack_token: "<your slack token>"
#  slack_channel: "<the slack channel>"
#  slack_signing_secret: "<your slack signing secret>"

#socketio:
#  user_message_evt: <event name for user message>
#  bot_message_evt: <event name for bot messages>
#  session_persistence: <true/false>
```

9. app.py - Simple Web Interface

```python
from flask import Flask, render_template, request, jsonify
import rasa.core.agent
from rasa.core.agent import Agent
import asyncio
import uuid

app = Flask(__name__)
```

```
agent = None


# Initialize the Rasa agent
def initialize_agent():
    global agent
    model_path = "./models"  # Update this path if needed
    agent = Agent.load(model_path)


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


@app.route('/webhook', methods=['POST'])
def webhook():
    try:
        data = request.get_json()
        user_message = data['message']
        sender_id = data.get('sender', str(uuid.uuid4()))


        # Use asyncio to handle the async nature of Rasa
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)
        responses = loop.run_until_complete(agent.handle_text(user_message,
sender_id=sender_id))
        loop.close()
        bot_responses = [response.get("text") for response in responses if response.get("text")]

        return jsonify({
            'status': 'success',
            'responses': bot_responses
        })

    except Exception as e:
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 500

if __name__ == '__main__':
    initialize_agent()
    app.run(debug=True, port=5005)
```

10. templates/index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>FAQ Chatbot</title>
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f5f5f5;
  }
  .container {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
  }
  .chat-container {
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    height: 500px;
    overflow-y: auto;
    padding: 20px;
    margin-bottom: 20px;
  }
  .message {
    margin-bottom: 15px;
    display: flex;
  }
```

```
.user-message {
  justify-content: flex-end;
}
.bot-message {
  justify-content: flex-start;
}
.message-content {
  max-width: 70%;
  padding: 10px 15px;
  border-radius: 18px;
}
.user-message .message-content {
  background-color: #007bff;
  color: white;
}
.bot-message .message-content {
  background-color: #e9ecef;
  color: #333;
}
.input-container {
  display: flex;
}
#user-input {
  flex-grow: 1;
  padding: 10px;
  border: 1px solid #ddd;
```

```
    border-radius: 4px;
    margin-right: 10px;
}
#send-button {
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
#send-button:hover {
    background-color: #0056b3;
}
.suggestions {
    display: flex;
    flex-wrap: wrap;
    margin-bottom: 15px;
}
.suggestion {
    background-color: #e9ecef;
    border: none;
    border-radius: 15px;
    padding: 5px 10px;
    margin: 5px;
    cursor: pointer;
```

```
        }
        .suggestion:hover {
            background-color: #d6d8db;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>FAQ Chatbot</h1>
        <div class="chat-container" id="chat-container">
            <div class="message bot-message">
                <div class="message-content">
                    Hello! I'm here to help with frequently asked questions. How can I assist you
today?
                </div>
            </div>
        </div>

        <div class="suggestions">
            <button class="suggestion" onclick="sendSuggestion('What is your return
policy?')">Return Policy</button>
            <button class="suggestion" onclick="sendSuggestion('How long does shipping
take?')">Shipping Info</button>
            <button class="suggestion" onclick="sendSuggestion('What products do you
offer?')">Products</button>
            <button class="suggestion" onclick="sendSuggestion('What payment methods do
you accept?')">Payment Options</button>
```

```
        <button class="suggestion" onclick="sendSuggestion('How can I contact
you?')">Contact Info</button>

      </div>



      <div class="input-container">

        <input type="text" id="user-input" placeholder="Type your question here...">

        <button id="send-button" onclick="sendMessage()">Send</button>

      </div>

    </div>



    <script>

      function addMessage(message, isUser) {

        const chatContainer = document.getElementById('chat-container');

        const messageDiv = document.createElement('div');

        messageDiv.className = isUser ? 'message user-message' : 'message bot-message';



        const contentDiv = document.createElement('div');

        contentDiv.className = 'message-content';

        contentDiv.textContent = message;



        messageDiv.appendChild(contentDiv);

        chatContainer.appendChild(messageDiv);



        // Scroll to bottom

        chatContainer.scrollTop = chatContainer.scrollHeight;

      }
```

```
function sendMessage() {
  const userInput = document.getElementById('user-input');
  const message = userInput.value.trim();

  if (message) {
    addMessage(message, true);
    userInput.value = '';

    // Send to backend
    fetch('/webhook', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        message: message
      })
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === 'success') {
        data.responses.forEach(response => {
          addMessage(response, false);
        });
      } else {
            addMessage("Sorry, I'm having trouble processing your request.", false);
          }
        })
        .catch(error => {
          console.error('Error:', error);
          addMessage("Sorry, I'm having trouble processing your request.", false);
        });
      }
    }

    function sendSuggestion(question) {
      document.getElementById('user-input').value = question;
      sendMessage();
    }

    // Allow sending message with Enter key
    document.getElementById('user-input').addEventListener('keypress', function(e) {
      if (e.key === 'Enter') {
        sendMessage();
      }
    });
  </script>
</body>
</html>
```

Setup and Running Instructions

1. Install dependencies:

   ```bash
   pip install -r requirements.txt
   python -m spacy download en_core_web_md
   ```

2. Train the model:

   ```bash
   rasa train
   ```

3. Run the chatbot:

   ```bash
   python app.py
   ```