E-BOOK: DESENVOLVENDO UM CHATBOT

Educacional para Treinamento de Funcionários na Vivo

vivo

SUMÁRIO

1. INTRODUÇÃO

Neste e-book, vamos guiá-lo através do processo de desenvolvimento de um chatbot educacional para o treinamento de funcionários na Vivo. Este projeto tem como objetivo ajudar a empresa a fornecer um treinamento mais eficiente e interativo, utilizando a tecnologia do ChatGPT.

2. Planejamento do Projeto

Antes de iniciar o desenvolvimento, é crucial planejar as funcionalidades e a arquitetura do sistema.

Funcionalidades Principais:

- Fornecer quizzes e perguntas sobre tópicos específicos.
- Responder perguntas dos funcionários com base em uma biblioteca de conhecimento.
- Fornecer feedback instantâneo sobre as respostas.
- Recomendar materiais de estudo adicionais.

Arquitetura:

- Backend em Python utilizando Flask.
- Integração com a API da OpenAI.
- Banco de dados para armazenamento de perguntas, respostas e materiais de estudo.
- Interface web simples para interação com o chatbot.

3. Configuração do Ambiente de Desenvolvimento

Ferramentas Necessárias:

- Python 3.8+
- Flask
- SQLite (ou PostgreSQL)
- Bibliotecas adicionais: requests, flask_sqlalchemy

Passos de Configuração:
1 - Instale o Python: Se ainda não tiver o Python instalado, baixe e instale a versão mais recente de python.org.

2 - Crie um ambiente virtual:

```
python -m venv venv
source venv/bin/activate  # Para Windows use `venv\Scripts\activate`
```

3 - Instale as dependências:

```
pip install flask flask_sqlalchemy requests
```

4. Estrutura do Projeto
Crie a estrutura de diretórios para o projeto:

```
chatbot-treinamento/
│
├── app/
│   ├── __init__.py
│   ├── models.py
│   ├── routes.py
│   └── templates/
│       └── index.html
├── config.py
├── run.py
└── requirements.txt
```

5. Criação da API com Flask
config.py:

```
import os

class Config:
    SECRET_KEY = os.getenv('SECRET_KEY', 'minha_chave_secreta')
    SQLALCHEMY_DATABASE_URI = 'sqlite:///chatbot.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

app/init.py:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

def create_app():
    app = Flask(__name__)
    app.config.from_object('config.Config')

    db.init_app(app)

    with app.app_context():
        from . import routes
        db.create_all()

    return app
```

run.py:

```
from app import create_app

app = create_app()

if __name__ == "__main__":
    app.run(debug=True)
```

6. Integração com a OpenAI API
app/routes.py:

```
from flask import request, render_template, jsonify
from app import db
import openai
import os

openai.api_key = os.getenv("OPENAI_API_KEY")

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

@app.route("/ask", methods=["POST"])
def ask():
    question = request.form["question"]
    response = openai.Completion.create(
        engine="text-davinci-003",
        prompt=question,
        max_tokens=150
    )
    return jsonify(response["choices"][0]["text"].strip())
```

7. Banco de Dados e Gerenciamento de Conteúdo
app/models.py:

```python
from . import db

class Quiz(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    question = db.Column(db.String(256), nullable=False)
    answer = db.Column(db.String(256), nullable=False)

class Material(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(256), nullable=False)
    content = db.Column(db.Text, nullable=False)
```

8. Desenvolvimento do Chatbot
Crie uma interface web simples para interação com o chatbot.
app/templates/index.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Chatbot Educacional</title>
</head>
<body>
    <h1>Chatbot Educacional</h1>
    <form id="chat-form">
        <input type="text" id="question" name="question" placeholder="Faça sua pergunta">
        <button type="submit">Enviar</button>
    </form>
    <div id="response"></div>
    <script>
        document.getElementById('chat-form').addEventListener('submit', function(event) {
            event.preventDefault();
            let question = document.getElementById('question').value;
            fetch('/ask', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ question: question }),
            })
            .then(response => response.json())
            .then(data => {
                document.getElementById('response').innerText = data;
            });
        });
    </script>
</body>
</html>
```

9. Implementação de Quizzes e Feedback

Adicione rotas para gerenciar quizzes e fornecer feedback.

app/routes.py:

```python
from flask import request, render_template, jsonify
from app import db
from app.models import Quiz, Material
import openai
import os

openai.api_key = os.getenv("OPENAI_API_KEY")

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

@app.route("/ask", methods=["POST"])
def ask():
    question = request.json["question"]
    response = openai.Completion.create(
        engine="text-davinci-003",
        prompt=question,
        max_tokens=150
    )
    return jsonify(response["choices"][0]["text"].strip())

@app.route("/quiz", methods=["GET"])
def get_quiz():
    quizzes = Quiz.query.all()
    return render_template("quiz.html", quizzes=quizzes)

@app.route("/quiz", methods=["POST"])
def submit_quiz():
    question_id = request.form["question_id"]
    answer = request.form["answer"]
    quiz = Quiz.query.get(question_id)
    if quiz.answer.lower() == answer.lower():
        return "Correto!"
    else:
        return "Incorreto! A resposta correta é: " + quiz.answer
```

```
                },
                body: JSON.stringify({ question: question }),
            })
            .then(response => response.json())
            .then(data => {
                document.getElementById('response').innerText = data;
            });
        });
    </script>
</body>
</html>
```

10. Testes e Implementação Final

Teste sua aplicação localmente para garantir que todas as funcionalidades estejam operando corretamente. Realize testes com diferentes cenários de uso, como responder quizzes e fazer perguntas variadas.

Dicas para Testes:

- *Verifique a responsividade da interface web.*
- *Teste a precisão das respostas fornecidas pelo ChatGPT.*
- *Assegure-se de que o feedback dos quizzes está correto.*
- *Revise a recomendação de materiais adicionais.*

Deploy: Considere implantar sua aplicação em uma plataforma como Heroku, AWS ou outra de sua preferência, para que possa ser acessada pelos funcionários da Vivo.

Conclusão: Parabéns! Agora você tem um chatbot educacional funcional que pode ser usado para treinar e educar os funcionários da Vivo de maneira eficiente e interativa.

- Mentoria completa
- Exercícios e desafios

vivo