Medical Symptom Checker - Complete Setup Guide

📋 Prerequisites

- Python 3.8 or higher
- Groq API key (get from https://console.groq.com)
- Modern web browser

📁 Project Structure

```
symptom-checker/
│
├── app.py                # Flask backend
├── index.html            # Frontend interface
├── requirements.txt      # Python dependencies
├── .env                  # Environment variables
└── symptom_checker.db    # SQLite database (auto-created)
```

🚀 Installation Steps

1. Create Project Directory

```bash
mkdir symptom-checker
cd symptom-checker
```

2. Create Virtual Environment

```bash
# Create virtual environment
python -m venv venv

# Activate on Windows
venv\Scripts\activate

# Activate on Mac/Linux
source venv/bin/activate
```

3. Install Dependencies

Create `requirements.txt` with the provided content, then:

```bash
pip install -r requirements.txt
```

4. Set Up Environment Variables

Create a `.env` file in the project root:

```bash
GROQ_API_KEY=your_actual_groq_api_key_here
```

Replace `your_actual_groq_api_key_here` with your real API key from https://console.groq.com/keys

5. Save All Files

- Save `app.py` (Flask backend)
- Save `index.html` (Frontend interface)
- Save `requirements.txt`
- Create `.env` file with your API key

▶️ Running the Application

Step 1: Start Backend Server

```bash
# Make sure virtual environment is activated
python app.py
```

You should see:

```
 * Running on http://127.0.0.1:5000
 * Debug mode: on
```

Step 2: Open Frontend

Open `index.html` directly in your browser, or serve it properly:

Option A - Direct File Open:

- Simply double-click `index.html`
- Or drag it into your browser

Option B - Local Server (Recommended):

```bash
# In a new terminal
python -m http.server 8000
```

Then visit: http://localhost:8000

🎯 Using the Application

1. **Enter Symptoms:**
   - Type your symptoms in the text area
   - Example: "I have a headache, fever of 101°F, and sore throat for 2 days"
2. **Analyze:**
   - Click "Analyze Symptoms"
   - Wait for AI analysis (5-10 seconds)
3. **View Results:**
   - See possible conditions
   - Get recommended next steps
   - Learn when to seek immediate care
4. **Check History:**
   - Click "View History" to see past queries
   - All queries are automatically saved

🔌 API Endpoints

POST /api/analyze

Analyze symptoms with AI

Request:

```json
```

```
{
 "symptoms": "headache, fever, sore throat"
}
```

Response:

json

```json
{
 "symptoms": "headache, fever, sore throat",
 "analysis": "Detailed AI analysis...",
 "timestamp": "2025-10-10T12:00:00"
}
```

GET /api/history

Get query history

Parameters:

- `limit` (optional): Number of records (default: 10)

Response:

json

```json
[
  {
    "id": 1,
    "symptoms": "headache, fever",
    "response": "Analysis...",
    "timestamp": "2025-10-10T12:00:00"
  }
]
```

GET /api/health

Health check endpoint

Response:

json

```
{
  "status": "healthy",
  "message": "Symptom Checker API is running"
}
```

🗄️ Database Information

SQLite Database

- **File:** `symptom_checker.db`
- **Auto-created:** On first run
- **Location:** Same directory as `app.py`

Schema

```sql
CREATE TABLE queries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    symptoms TEXT NOT NULL,
    response TEXT NOT NULL,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
)
```

Viewing Database

```bash
# Open SQLite CLI
sqlite3 symptom_checker.db

# View all queries
SELECT * FROM queries;

# Exit
.quit
```

🛠️ Troubleshooting

CORS Errors

Problem: Frontend can't connect to backend

Solutions:

- Ensure backend is running on port 5000
- Check `API_URL` in `index.html` matches backend
- Clear browser cache

API Key Errors

Problem: "Error analyzing symptoms: Invalid API key"

Solutions:

- Verify `.env` file exists in project root
- Check API key is correct (no extra spaces)
- Restart backend after changing `.env`
- Get new key from https://console.groq.com/keys

Database Locked

Problem: "Database is locked"

Solution:

```bash
# Delete database and restart
rm symptom_checker.db
python app.py
```

Module Not Found

Problem: "ModuleNotFoundError: No module named 'flask'"

Solution:

```bash
# Ensure virtual environment is activated
pip install -r requirements.txt
```

Port Already in Use

Problem: "Address already in use"

Solution:

```bash
# Find and kill process on port 5000
# Windows
netstat -ano | findstr :5000
taskkill /PID <PID> /F

# Mac/Linux
lsof -ti:5000 | xargs kill -9
```

⚙️ Customization

Change LLM Model

Edit `app.py`:

```python
model="llama-3.1-70b-versatile"  # Change to other Groq models
# Options: llama-3.1-8b-instant, mixtral-8x7b-32768, gemma-7b-it
```

Adjust Response Length

```python
max_tokens=1024  # Increase for longer responses (max: 4096)
```

Change API Port

```python
app.run(debug=True, port=5000)  # Change port number
```

Remember to update `API_URL` in `index.html`!

Database Location

```python
conn = sqlite3.connect('path/to/your/database.db')
```

🔒 Security Best Practices

For Development

- ✅ Never commit `.env` file
- ✅ Add `.env` to `.gitignore`
- ✅ Use environment variables for secrets
- ✅ Keep API keys private

For Production

- 🔐 Use PostgreSQL instead of SQLite
- 🔐 Add user authentication
- 🔐 Implement rate limiting
- 🔐 Use HTTPS only
- 🔐 Add input validation & sanitization
- 🔐 Set up proper CORS policies
- 🔐 Use production WSGI server (Gunicorn)
- 🔐 Enable logging and monitoring

🚢 Production Deployment

Using Gunicorn (Recommended)

```bash
pip install gunicorn
gunicorn -w 4 -b 0.0.0.0:5000 app:app
```

Docker Deployment

```dockerfile
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "app:app"]
```

Environment Variables for Production

```bash
FLASK_ENV=production
GROQ_API_KEY=your_key
DATABASE_URL=postgresql://...
```

📚 Additional Resources

- **Groq Documentation:** https://console.groq.com/docs
- **Flask Documentation:** https://flask.palletsprojects.com/
- **SQLite Documentation:** https://www.sqlite.org/docs.html
- **Python Virtual Environments:** https://docs.python.org/3/library/venv.html

🎓 Learning Resources

Understanding the Code

- **Flask Basics:** RESTful API design
- **Groq API:** LLM integration patterns
- **SQLite:** Database management
- **CORS:** Cross-origin requests
- **JavaScript Fetch:** Async API calls

📝 Testing the API

Using cURL

```bash
# Test health endpoint
curl http://localhost:5000/api/health

# Test symptom analysis
curl -X POST http://localhost:5000/api/analyze \
  -H "Content-Type: application/json" \
  -d '{"symptoms": "headache and fever"}'

# Test history
curl http://localhost:5000/api/history
```

Using Postman

1. Import endpoints into Postman
2. Set method and URL
3. Add JSON body for POST requests
4. Send and view responses

💡 Feature Ideas

Want to extend the project? Consider adding:

- User authentication and profiles
- Export results as PDF
- Email notifications
- Multi-language support
- Voice input for symptoms
- Integration with health APIs
- Symptom tracking over time
- Medical resource recommendations

🐛 Known Issues

- SQLite not suitable for high concurrency
- Frontend needs refresh for new history
- No real-time updates
- Limited error recovery

✅ Success Checklist

- [ ] Virtual environment created and activated
- [ ] All dependencies installed
- [ ] `.env` file created with valid API key
- [ ] Backend running on port 5000
- [ ] Frontend opens without errors
- [ ] Can submit symptoms and get analysis
- [ ] History feature works
- [ ] Database file created automatically

🆘 Getting Help

If you encounter issues:

1. Check the troubleshooting section
2. Verify all files are saved correctly
3. Ensure virtual environment is activated
4. Check console/terminal for error messages
5. Verify API key is valid and active