System Fault Reporter

Guided chatbot for structured fault reporting and escalation

Abstract

A chatbot assistant that guides users through the structured process of reporting a system fault. It collects key technical details (subsystem, severity, and description), confirms the report, and outputs a complete fault summary for downstream tracking or escalation.

Table of Contents

# Export Instructions

This model not only defines the chatbot logic but also serves as a demonstration of **efficient documentation workflows**.  
Its purpose is to show how both chatbot behavior and user-facing documentation can be generated from a **single source of truth** — the structured Markdown definition itself.

Thanks to this unified format, you can generate various output formats (such as .docx) directly from this file without duplication or inconsistency.

You can export the documentation in two ways using Pandoc:

### 1. Export only the User Documentation (filtered)

Use a Lua filter (e.g., extract\_userdoc.lua) to extract just the user documentation: ### 1. Export only the User Documentation (filtered)

Use a Lua filter (e.g., extract\_userdoc.lua) to extract just the user documentation:

# Export only the info section to Word (.docx)  
pandoc system\_fault\_reporter.md --lua-filter=extract\_info.lua --toc --toc-depth=3 -o info\_only.docx  
  
# Export the complete document to Word (.docx)  
pandoc system\_fault\_reporter.md --toc --toc-depth=3 -o complete.docx  
  
# Export the info section as a slide presentation (e.g., for Beamer, reveal.js, or PowerPoint)  
pandoc system\_fault\_reporter.md --slide-level=2 --toc --toc-depth=2 --reference-doc=layout.pptx --lua-filter=extract\_info.lua -o info\_only.pptx

### 2. Export the Complete Documentation

To export the full Markdown file as a .docx with a table of contents:

pandoc system\_fault\_reporter.md --toc --toc-depth=3 -o complete\_doc.docx

You can also apply a custom reference template using --reference-doc=layout.docx if desired.

# System

The chatbot helps engineers report system faults by collecting structured input.  
It first asks which subsystem is affected, then requests the severity level and a short description of the issue.  
After gathering all required fields, it summarizes the fault report and asks for final confirmation.

You are a technical assistant who helps users file structured system fault reports.   
1. First, ask for the affected subsystem (e.g., networking, power, storage).   
2. Then, ask for the severity level (e.g., critical, warning, info).   
3. Ask for a short description of the issue.   
4. If any field is missing, keep asking until all required details are provided.   
5. Once all data is available, present a complete summary of the fault report.   
6. Ask the user to confirm the report submission.   
7. If confirmed, output the full report.   
8. If not confirmed, respond with a polite follow-up message.

# Schema

The schema defines the expected structure of a fault report interaction.

from pydantic import BaseModel, Field   
from typing import Optional   
  
class Schema(BaseModel):   
 subsystem: Optional[str] = Field(None, description="The system component affected by the fault.")   
 severity: Optional[str] = Field(None, description="Severity level of the fault (e.g., critical, warning).")   
 description: Optional[str] = Field(None, description="Brief summary of the issue.")   
 confirmation: Optional[bool] = Field(None, description="Whether the fault report was confirmed by the user.")   
 response\_to\_user: Optional[str] = Field(None, description="Message to the user if the report was not confirmed.")

# Response

This response template provides either a detailed fault report or a follow-up message.

% if RESPONSE["confirmation"]:   
🛠 \*\*Fault Report Submitted:\*\*   
- \*\*Subsystem:\*\* ${RESPONSE["subsystem"]}   
- \*\*Severity:\*\* ${RESPONSE["severity"]}   
- \*\*Description:\*\* ${RESPONSE["description"]}   
  
✅ Your report has been logged. Thank you for your input.   
% else:  
💬 ${RESPONSE.get("response\_to\_user", "Would you like to update or confirm your report?")}  
% endif

# User Documentation

## 1. Overview

This chatbot guides users through reporting a technical system fault.  
It collects structured inputs — the affected subsystem, severity level, and a brief description — and generates a complete fault report for escalation or tracking.

This slide introduces the chatbot and its purpose in assisting with structured fault reporting.

## 2. Interaction Flow

1. **Select Subsystem** — User specifies the affected area (e.g., networking, power).
2. **Set Severity** — User selects how critical the issue is.
3. **Describe Issue** — A short explanation of the fault is provided.
4. **Confirm Report** — The chatbot summarizes the details and asks for confirmation.
5. **Submit or Revise** — Based on confirmation, the bot submits the report or follows up politely.

This slide outlines the steps the user follows during the interaction with the chatbot.

## 3. Input Fields

| Field | Description | Example |
| --- | --- | --- |
| subsystem | The affected system area | networking |
| severity | The level of urgency | critical |
| description | A short explanation of the problem | "Server offline" |
| confirmation | Indicates if the report is ready for submission | true |
| response\_to\_user | Message shown when the report isn’t confirmed | "Please confirm" |

This slide details each of the fields the chatbot collects from the user.

## 4. Output Behavior

* If all required fields are present and confirmation is true, a full fault report is generated.
* If not confirmed, a helpful message is shown and the bot remains ready to proceed.

This slide explains how the chatbot behaves based on user input.

## 5. Usage Notes

* The bot ensures all required fields are filled before submission.
* It provides concise prompts for clarity.
* Useful in helpdesk, monitoring, or incident reporting contexts.

This slide offers general tips and context for effective use of the chatbot.