Software User Manual

Insurance Assistant

Pouriya Miri

Dec 24th, 2025

Contents

1 Description **2**

2 Prerequire **2**

3 Running the Application **2**
- 3.1 Installation . . . 2
- 3.2 Configuration . . . 3
- 3.3 Running Interfaces . . . 3

4 Using the Application (End-User Flow) **4**
- 4.1 Getting Policy Information . . . 4
- 4.2 Getting a Premium Estimate . . . 4
- 4.3 Reporting a Claim . . . 5
- 4.4 Ending the Call or Speaking to a Human . . . 5

5 Features, Technologies and Options **6**
- 5.1 Features . . . 6
- 5.2 Technologies and Libraries . . . 6

6 How It Works (Architecture) **7**
- 6.1 High-level Description . . . 7
- 6.2 Components . . . 8
- 6.3 Differences between `app.py` and `app1.py` . . . 9

7 Improvements and Additional Functionalities **9**
- 7.1 Code Quality and Architecture . . . 9
- 7.2 User-Facing Features . . . 10

1 Description

This application is a virtual assistant for car insurance customers in Slovenia. It can work in two ways: by text in the terminal, or by voice using a microphone and speakers. The assistant can answer questions about insurance policies, help calculate an estimated insurance price, and guide users through reporting a car accident. A typical user would be a driver seeking information about policies or wishing to report an accident. The tool could also be used by insurers to prototype virtual agents for call-centres or self-service kiosks.

2 Prerequire

The application is a Python project that relies on several third-party libraries. Before installing it, you need the following:

- Operating system: any desktop platform capable of running Python 3.8+ (Linux, Windows or macOS). For the voice version, the machine must have a microphone and speakers.
- Python runtime: version 3.8 or higher with the ability to create virtual environments.
- Package manager: *pip* to install dependencies.
- Audio drivers: *sounddevice* and *soundfile* require native audio drivers to capture and play audio. On Linux, the *portaudio* backend must be available.
- Internet access: $edge - tts$ downloads synthesized speech from Microsoft's Edge Text-to-Speech service.
- Reference documents: the assistant builds a retrieval index from the files in the *docs/* folder. These files include policy FAQs and claim instructions, such as claim reporting steps and premium-calculation methodology.

3 Running the Application

3.1 Installation

Follow these steps to install the application:

- Obtain the code: download and unzip the project or clone the repository [1].

```
git clone https://github.com/PouriyaMiri/Insurance_Agent
cd Insurance_Agent
```

[1] https://github.com/PouriyaMiri/Insurance_Agent

- Create a virtual environment *(OPTIONAL)*: open a terminal in the project directory and run:

```
python -m venv .venv
source .venv/bin/activate    # Linux/macOS
.venv\Scripts\activate       # Windows
```

- Install dependencies: install the required libraries using *pip*:

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

3.2 Configuration

The assistant uses sensible defaults but can be configured in several ways:

- **Document folder:** `app_common.build_context(docs_path='./docs')` reads `.txt` and `.md` files from the `docs/` folder and builds an in-memory index. To supply your own documents, copy them into this folder or change the `docs_path` argument when building the context.
- **Exit phrases:** The list of words that end the call (e.g., "hang up", "goodbye", "stop") can be modified in `app_common.EXIT_PHRASES`.
- **Voice settings:** The `VoiceOut` class accepts `rate` (words per minute), `volume` (0–1.0) and `voice` (e.g., `en-US-JennyNeural`) parameters. You can change these when constructing the `VoiceOut` instance in `app1.py`.
- **Voice activation detector (VAD):** In the voice version, thresholds such as `start_threshold` and `stop_threshold` control when recording begins and ends. These values can be tuned in `app1.py` when instantiating `VADRecorder`.
- **Whisper model size:** The `WhisperSTT` class accepts a `model_size` argument (small, medium, etc.) and a `device` (e.g., `cpu` or `cuda`).
- **Environment variables:** Setting `VOICEOUT_ECHO_ON_FAIL=1` will print the assistant's response to the console if audio playback fails.

There are no explicit command-line flags; configuration is performed by editing the Python files or using environment variables.

3.3 Running Interfaces

Depending on how you want to interact with the assistant you have two entry points:

- **Text console interface:** Run

```
python app.py
```

The program prints a greeting and waits for your input. Type your question or instruction and press Enter. To end the call type any of the exit phrases (e.g., “bye”, “quit”).

- **Voice interface:** Run

  ```
  python app1.py
  ```

  The assistant speaks the greeting using text-to-speech. It then listens for speech: begin speaking after the tone. When you finish your sentence, the system transcribes your words using Whisper and prints them for confirmation. The agent responds via synthesized voice. To interrupt a response, start speaking again (barge-in). Say an exit phrase like “goodbye” or “hang up” to end the session.

Both interfaces build the document index on startup, so the first run may take a few seconds while embeddings are generated.

4 Using the Application (End-User Flow)

4.1 Getting Policy Information

If you ask about coverage, deductibles, or exclusions, the assistant consults the documents in `docs/` to return a concise answer. For example: **User:** “Tell me about premium coverage.”
Agent: “Premium: broadest cover (liability + collision + theft/fire/weather) and optional add-ons like roadside assistance.” These summaries are derived from the policy FAQ.

You can follow up with clarifying questions such as “What are the differences between basic and standard?” and the assistant will compare the coverage levels or ask if you want the cheapest option.

4.2 Getting a Premium Estimate

To obtain a quote, tell the assistant you need a price or say “I need car insurance”. The agent will guide you through the required inputs:

- **Vehicle age:** Either give the model year (e.g., “2015”) or state the age in years. The assistant will convert the year into age automatically.
- **Engine power:** Supply the horsepower (e.g., “100 hp”) or provide engine size in litres (e.g., “1.4 L”). If you provide the engine size, the system estimates horsepower from it.
- **City:** State where the vehicle is primarily used (Ljubljana, Maribor, Celje, Koper).
- **Coverage level:** Choose basic, standard or premium. The agent can also infer “cheapest” as basic.

Once all information is provided, the assistant calculates the monthly premium using a simple formula. According to the internal methodology document, the estimate depends on a base rate, vehicle age, engine power, city risk factor and coverage level. The result is presented as "€ X per month" and is clearly marked as an estimate. You can ask to compare all three coverage levels. The agent will call the premium calculator for each level and present a table of basic, standard and premium monthly rates.

4.3 Reporting a Claim

When you say "I want to file a claim" or mention an accident, the system starts a claim intake. It collects the following information one by one:

- **Policy number:** Read your insurance or policy number aloud or type it in.
- **Injuries:** Confirm whether anyone was injured; if yes, it advises you to contact emergency services first.
- **Accident location:** Give the city and, optionally, a local area.
- **Accident date:** Say "today", "yesterday" or the calendar date (YYYY-MM-DD).
- **Description:** Briefly describe what happened.
- **Police report:** State whether the police were notified and, if applicable, provide the report reference number.
- **Vehicle condition:** Indicate if the car is drivable.
- **Third-party involvement:** State whether other vehicles were involved.

After collecting all required fields, the assistant generates a unique claim number, summarizes the details and suggests next steps: contacting emergency services if needed, taking photos, collecting witness contacts and saving receipts. A reminder that claims should typically be reported within 24 hours is also included.

4.4 Ending the Call or Speaking to a Human

At any time, you can end the conversation by typing or saying an exit phrase such as "hang up", "quit" or "stop". If you explicitly ask for a human agent (e.g., "I want to talk to a person"), the system stops the automated flow and simulates a transfer to human support.

5 Features, Technologies and Options

5.1 Features

- **Conversational assistant:** Interacts in natural language to answer questions about car-insurance coverage and procedures.
- **Retrieval-augmented QA:** Uses a small retrieval index to answer FAQs about policy terms, claims processes and exclusions. The answers are drawn from the bundled documents.
- **Premium estimation:** Collects basic vehicle and coverage details and computes an estimated monthly premium using a simple model and risk factors.
- **Claim intake:** Guides the caller through reporting an accident by capturing policy number, injuries, location, date, description, police involvement and vehicle condition, then generates a claim number and provides next steps.
- **Coverage comparison:** Compares monthly premiums across basic, standard and premium options.
- **Digit recognition and normalisation:** Interprets spoken digits, engine sizes and city names (with fuzzy matching for Slovenian city variants).
- **Voice interface:** Optional speech-to-text (Whisper) and text-to-speech (Edge TTS) allow hands-free operation. Users can barge-in during playback to interrupt long responses.
- **Human handoff:** Responds to requests for a human operator by ending the automated session.

5.2 Technologies and Libraries

Programming language: Python 3.8+.
Frameworks/libraries:

- **Rich:** Renders nicely formatted panels and coloured text in the console.
- **Pydantic:** Models session state, intent results and premium results.
- **Sentence-Transformers & FAISS:** Embed documents and build a retrieval index for question answering.
- **Faster-Whisper:** Runs OpenAI's Whisper speech-to-text model for voice input.
- **Edge-TTS:** Synthesizes responses using Microsoft's neural voices.

- **SoundDevice and SoundFile:** Capture audio from the microphone and play back generated speech.
- **NumPy:** For audio processing and vector calculations.

External services: The TTS module uses Microsoft's Edge TTS service; Whisper STT downloads pre-trained models; no external APIs are called for the Q&A or premium calculation.

6 How It Works (Architecture)

6.1 High-level Description

At startup the assistant builds a retrieval index from the text files in the `docs/` directory. The index is created by splitting each document into overlapping chunks, embedding them using a pre-trained sentence transformer and adding them to a FAISS index. The conversation state is maintained in a `SessionState` object containing user-provided slots (e.g., city, vehicle age), the last detected intent and counters.

For each user turn the system performs the following steps:

- **Input:** The console version reads a line of text; the voice version records audio until silence is detected and then runs Whisper to transcribe the utterance.
- **Natural-language understanding:** The `nlu.py` module detects the intent (document Q&A, claim reporting, premium estimate, coverage comparison or human handoff) and extracts entities such as coverage level, horsepower or city.
- **Session update:** Extracted entities are stored in the session state. The manager also derives the vehicle age from a provided year, normalises city names and checks for exit phrases.
- **Dialogue logic:** Based on the intent and current state, the `dialogue_manager()` decides what to do:
  - **Document Q&A:** Build a query and retrieve the top document chunks. If the user appears dissatisfied or asks for differences, follow-up questions are generated. The retrieval results are summarised to form the answer.
  - **Premium estimation:** Ask for any missing details (age, power, city, coverage). When all slots are filled, call `calculate_premium()` which applies factors for age, horsepower, city and coverage level to compute a monthly premium. Results include a breakdown of factors.

  - **Claim intake:** Set a flag to indicate that a claim is being reported. Repeatedly ask for missing fields until all are provided, then generate a unique claim number and present the summary with next-step guidance.
  - **Coverage comparison:** Call the premium estimator for each coverage level and list the monthly costs.
  - **Human handoff or exit:** Immediately terminate the conversation and send a handoff message.

- **Output:** The assistant returns a `TurnResult` containing the response text and a flag to end the call. The console version prints this text in a Rich panel; the voice version speaks it using Edge-TTS and allows barge-in.

6.2 Components

- **Document indexer (`rag.py`):** Manages the embedding model and FAISS index; splits documents into chunks and handles retrieval requests.
- **Natural-language understanding (`nlu.py`):** Contains intent classification and entity extraction heuristics based on keyword matching. Synonyms for coverage levels are mapped to canonical values.
- **Dialogue manager (`dialogue.py`):** Orchestrates the conversation. It uses helper functions to normalise numbers, parse dates and yes/no answers, estimate horsepower from engine size and manage claim and quote flows.
- **Premium calculator (`premium.py`):** Defines a simple pricing formula with adjustable factors for vehicle age, horsepower, city and coverage level. The methodology and terms are documented in the premium calculation file.
- **Voice recorder (`voice_loop.py`):** Implements a basic voice activation detector that starts recording when the sound level exceeds a threshold and stops after silence. It saves the captured audio to a temporary `.wav` file. It also wraps Whisper for transcription.
- **Voice output (`voice_out.py`):** Wraps Microsoft Edge TTS into a synchronous API. It downloads speech to a temporary file and streams it through `sounddevice`. A stop event allows barge-in and interruption of ongoing playback.
- **Console utilities (`app_common.py`):** Sets up the Rich console, defines exit phrases and builds the context (document index and session state).

During a typical premium-estimate call the data flows from the microphone through voice activation and Whisper to text, passes through intent detection and entity extraction, triggers premium calculation, and returns back through text-to-speech for the user.

6.3 Differences between `app.py` and `app1.py`

- **Interface:** `app.py` implements a simple text console interface. It greets the user, reads input via the keyboard and prints responses using Rich panels. `app1.py` adds a voice layer: it uses a voice activation detector to record utterances, transcribes speech with Whisper and speaks responses using the Edge TTS service.
- **User experience:** The console version is synchronous—you type a question and wait for a printed answer. The voice version runs several threads to handle audio recording and playback concurrently. Users can barge-in during a long reply by speaking again, which stops the current TTS playback.
- **Dependencies:** `app1.py` depends on additional modules (`voice_loop.py` and `voice_out.py`) and audio hardware. `app.py` works in any terminal without requiring a microphone or speakers.
- **Entry points:** Both files call `build_context()` to load documents and then invoke `dialogue_manager()` on each user turn. `app.py` loops on text input, while `app1.py` loops on audio files produced by the voice recorder and uses a queue to process them.
- **Naming:** `app1.py` could be more descriptive (e.g., `voice_app.py`) to clarify that it is the voice-enabled version. Both files share common logic through the dialogue and `app_common` modules.
- **Which to run:** Users wishing to interact via text should run `app.py`. Those testing the voice experience should run `app1.py` provided audio hardware and the additional dependencies are available.

7 Improvements and Additional Functionalities

7.1 Code Quality and Architecture

- **Centralised configuration:** Move hard-coded parameters (exit phrases, VAD thresholds, pricing factors) into a configuration file or environment variables. Use a settings class to validate and load these values.
- **Error handling and logging:** Wrap audio, TTS and STT calls with structured error handling and log unexpected exceptions. Use Python's logging module instead of printing to the console.
- **Asynchronous architecture:** Refactor the voice version to use Python's `asyncio` throughout. This would simplify concurrency instead of manually spawning threads and queues.
- **Testing:** Add unit tests for NLU functions, premium calculation and claim parsing. Mock audio and network calls to test the voice pipeline.

- **Input validation:** Improve validation of numeric inputs (e.g., years, horsepower) and handle out-of-range values gracefully. Provide user feedback when inputs are invalid rather than silently ignoring them.
- **Modularisation:** Separate the dialogue logic into distinct classes or functions for Q&A, claims and quotes. This would make it easier to extend or replace individual modules.
- **Security:** Externalise API keys and secrets (e.g., for Edge TTS) using environment variables; never commit keys to source control. Sanitize all user input before processing.

7.2 User-Facing Features

- **Persistent sessions:** Allow users to resume a conversation later by storing claim or quote details in a database or file.
- **Multi-language support:** Add additional voices and language models (e.g., Slovenian) and detect user language automatically.
- **Send results via email or SMS:** After generating a quote or claim number, ask for contact information and deliver a summary through email or text.
- **Additional underwriting factors:** Incorporate driver age, accident history, mileage and vehicle value into the premium calculation to produce more accurate estimates.
- **Enhanced RAG content:** Include more detailed policy documents and external knowledge bases to answer a wider range of questions about insurance terms, deductibles and exclusions.
- **Web or mobile interface:** Wrap the assistant into a web application or mobile app to reach a broader audience. This could reuse the same backend but provide a graphical interface.
- **Real handoff integration:** Integrate with a call-centre system to actually transfer the caller to a human representative when requested, instead of simulating the transfer.

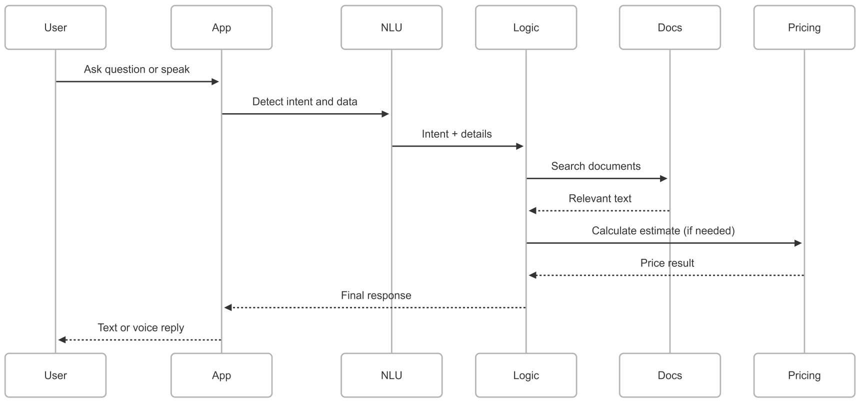

Figure 1: Sequence Diagram