Checklist Execution Assistant

Checklist Execution Assistant Voice-enabled Checklist readout and verification assistant. It finds a checklist based on user's voice command, reads items from the checklists, waits for user to confirm every item and verifies item has been actually executed.

Voice interaction is handled by Alexa engine. It translates voice commands to requests for backend, which tracks which checklist and checklist item is being executed at the moment. Backend sends back result to Alexa and it is spoken to the user. There is also a web page that lists all checklists registered in the system and shows current state of execution. Items that are validated by the system have green text, items that are checked by the user but are not validated by the system have red text and successfully checked and verified items have green background.

Technologies and Framework

The demo was developed using the following technologies and libraries:

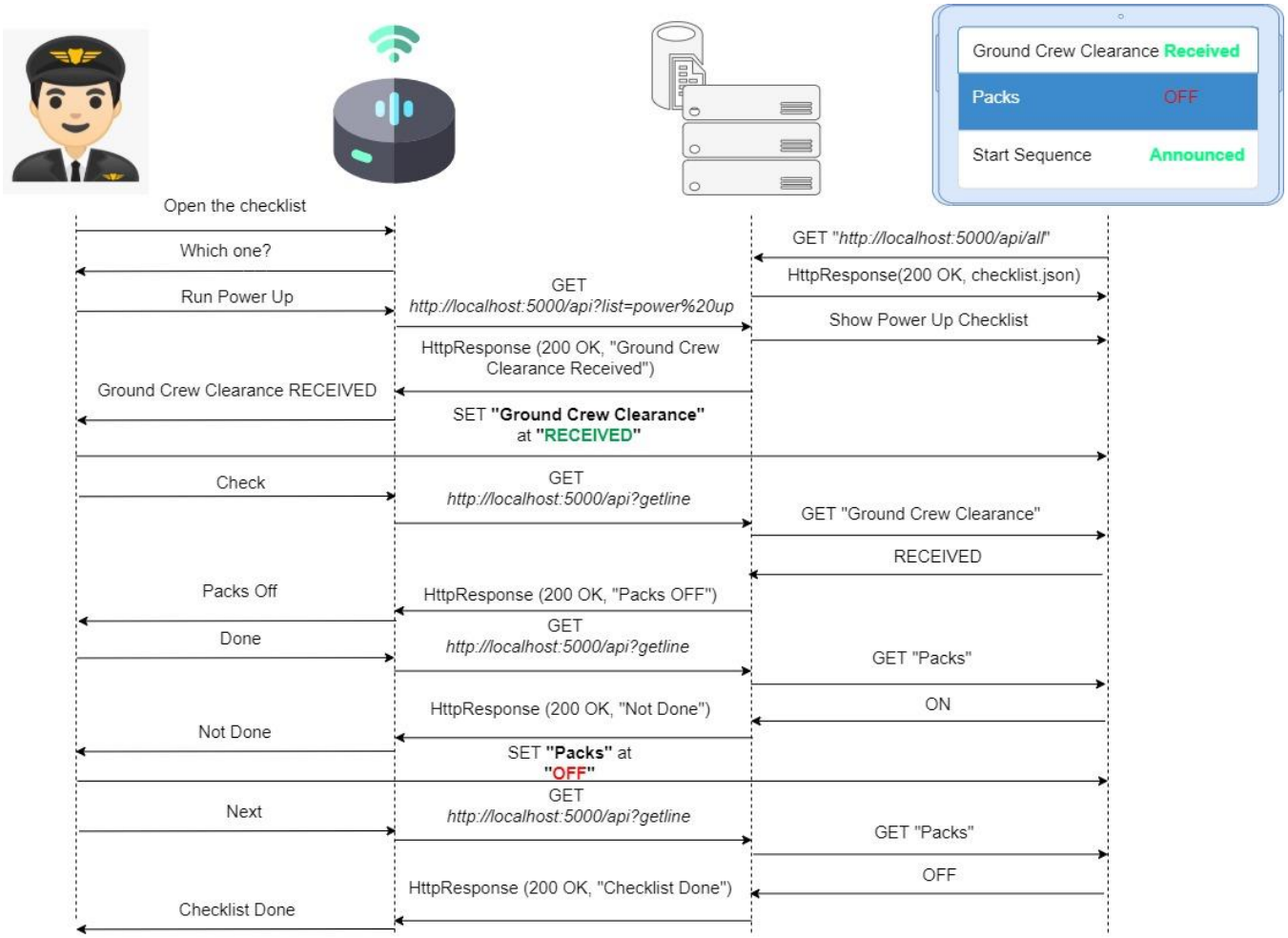
- Voice Assistant: AMAZON ALEXA (Python)
- Backend (Python)
  - Flask: https://www.palletsprojects.com/p/flask/
  - Flask Sockets: https://github.com/heroku-python/flask-sockets
  - JSON: checklist file
- GUI: it simulates the display screen in the cockpit
  - Bootstrap
  - CSS
  - HTML
  - Javascript

Demo

The demo is based on the interaction of 4 elements:

- Pilot: asks for a specific checklist and runs its instructions
- Alexa: waits for a specific command (**open the checklist**) and replies to the pilot with the checklist's instructions
- Server: creates a web service for providing at Alexa and at GUI the access to the checklists
- GUI: is a web page to simulate the pilot's display where the checklists are showed and the pilot can check its values.

Generally, the device, which will show the instructions, will load all checklists during the aircraft's pre-flight configuration phase. For this reason our GUI queries the server for getting all checklists.



Alexa

Alexa is the Amazon voice assistant and it is the heart of this demo. It manages two different communications:

- Pilot communication: Alexa receives well defined voice commands to understand and process them. If the voice commands are understandable for Alexa, it will reply with particular instructions for the pilot, otherwise it will ask for something more understandable.

- Server communication: once received the right commands from the pilot, Alexa needs to get the checklists from the server or from some repository where the checklists are stored. This communication is made up of GET requests over HTTP using the service provided by the server. For each GET request, the server will reply at Alexa with a string which will be provided to the pilot as response at his voice command.

The interaction between Alexa and Pilot is managed implementing a new skill for Alexa. The skill is a ability to understand particular topic such as playing music, providing weather forecast and so on. In this demo Alexa has been provided with a new custom Python skill for reading the checklist's instructions and providing them to the pilot in according to his request.

Each skill is defined by two main components:

- Intent: is the pilot's request. In this demo there are only two intents:

    - ChecklistIntent: manages the requests for opening and reading the first instruction of a checklist. This intent has got a *Slot* which is a parameter for adding different values which will be processed by Alexa. Its slot is *"checklist"* which values are the names of the checklists available in the server. The server checklist available for the demo are *"Engine Failure", "Power Up"*

    - AMAZON.NextIntent: is a built-in Amazon intent for getting the following instruction. Once the pilot got the first instruction, he can ask for the following ones through this built-in intent.

- Utterance: identifies the keyword to run that particular intent. Inside each utterance is possible defines multiple keywords in combinations with the slots. For example, for running the checklist "Power Up", the pilot can ask: "*RUN POWER UP"*. Run is the keyword and Power Up is the slot's value. In the next page there are the main utterances for each intent.

Checklist Slot	ChecklistIntent	Amzazon.NextIntent
Power UP	Run Power Up Start the Power Up	Check Next
Engine Failure	Run Engine Failure Start the Engine Failure	Check Next

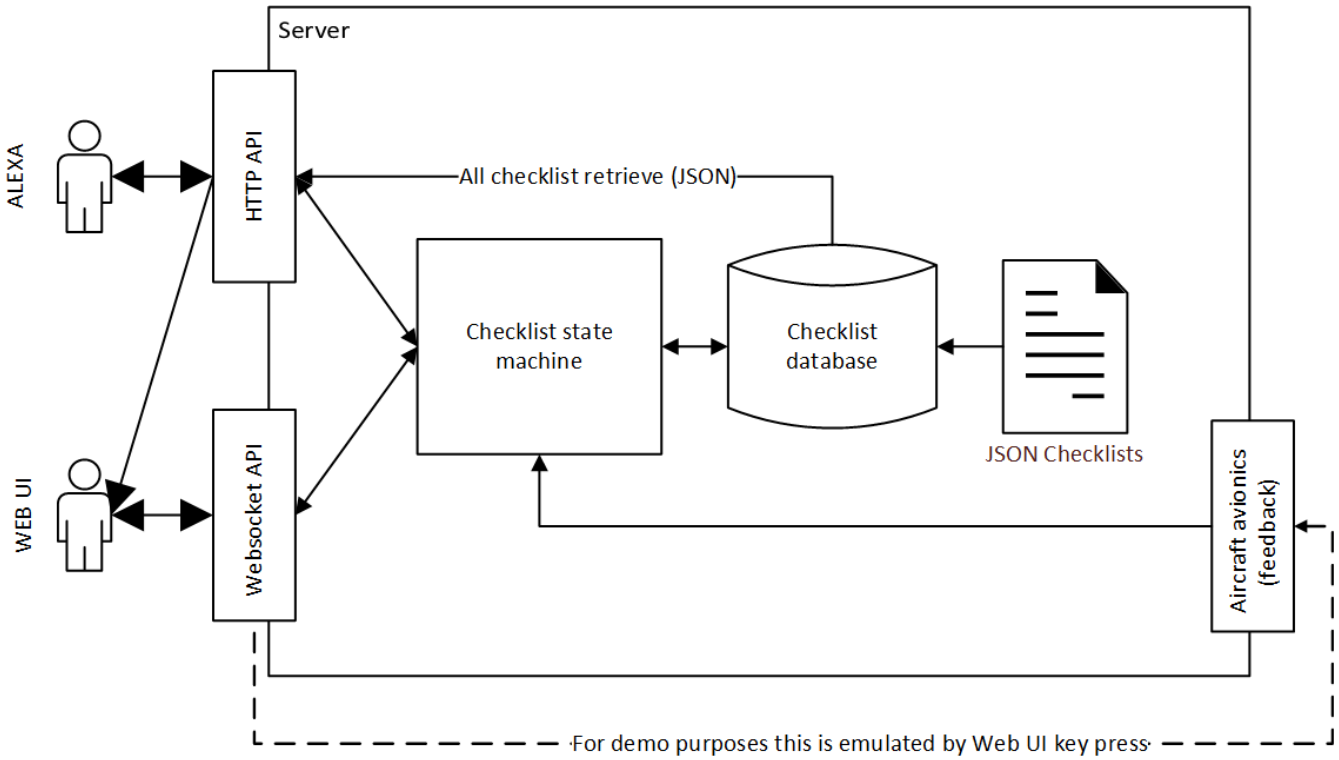
How does the Alexa work?

1. Pilot asks: *"Open the checklist"* ➔ it's identified from Alexa as LaunchRequest to start the skill.
2. Alexa replies *"Which one?"* ➔ it's the result of LaunchRequest.
3. Pilot says *"Run/Start Power Up/Engine Failure"* ➔ It's the utterance for the ChecklistIntent. Alexa process it into ChecklistIntentHandler:
   - Get the name of the checklist from the pilot's voice command
   - Make a GET Request at http://localhost:5000/api?list=checklistname
   - If Alexa get a success response, it submits another GET request for collect the first instruction of that checklist at http://localhost:5000/api?getline
   - Alexa replies to the pilot with the first instruction of the target checklist
   - In case of any failure, Alexa reports an error message to the Pilot and asks for another command.
4. Once pilot run the instructions, he asks *"Check/Next"* for getting the next instruction ➔ Alexa recognize the utterance "Check/Next" and manages it as Amazon.NextIntent into NextIntentHandler. In this handler, Alexa makes only a GET request for http://localhost:5000/api?getline to retrieve the next instruction.

5. Alexa replies to the pilot with the response received from the server which can be:
    - Next instruction
    - Error message: *"Not Done"* ➔ it means the pilot asks for new instruction jumping the execution of the previous one
    - *"Checklist Done"* ➔ it means the checklist has been completed successfully.
6. If the pilot receives the next instruction, he can run it and then asks again for the next one (point 4)

Server

Server is state keeper. It's designed as single user and therefore it keeps only 2 internal states – selected checklist and selected line. It has 3 interfaces – http API addresses (http://localhost:5000/api and http://localhost:5000/api/all), websocket interface supporting single connected user and Aircraft avionics feedback which is now emulated via websocket API. Whole server functionality is described below.



Checklist state machine has following logic:

1. User is connected via Web UI. Page retrieves from /api/all all checklists and open websocket to server.
2. User requests list (/api?list=list name)
    a. List is found and opened
    b. Error is reported
3. If list is opened, first line is automatically read and also list is shown in Web UI with indication on first line.
4. User confirms line by saying “Check”
    a. Alexa reports it back to server and it must decide if it was successful or not.
    b. If line is automatic, we don’t have feedback. Therefore we trust user and go to next line.
    c. If line is verifiable via feedback (**the user press Q button on keyboard**), we are verifying if action really happened.
        i. If it happened, we mark line as done in Web UI via websocket and go to next line. Therefore Alexa receives next instruction to be read.
        ii. If not, Alexa receives “Not done” message and Web UI receives error for given line.
    d. It will either wait for keyword “Check” if line was not done or proceed to next line.
5. This loops until all instructions are done. At end of list Alexa informs user about competition.

GUI – Graphical User Interface

Web page GUI simulates airplane avionics system that is able to display checklists and their execution to the pilot.

Web page is opened in web browser using URL http://localhost:5000/. When page loads, it fetches all checklists from backend using AJAX call to http://localhost:5000/api/all. Checklists names are shown in the left menu grouped into two groups: “Standard Checklists” and “Emergency Checklists”. Each checklist item is clickable and on click, it displays checklist items in the center part of the page.

Web page also opens a websocket to the backend server through which it receives updates of the state of the Checklist state machine. The message sent via web socket from backend to web page is a JSON message and it can be one of the two types:

- {"listid": <checklist_id>} – indicates the ID of the checklist user wants to open. Web page finds the checklist and shows its items similarly to when user manually clicks on a checklist name.
- {"itemid": <checklist_item_id_or_status>} – indicates execution state of the current checklist item:
    - When <checklist_item_id> is received it means the current item has been checked and successfully verified if the item is automatically verifiable and it changes background to green. The received ID is the ID of the next item.
    - When “fail” is received it means that the current item has been checked but it failed to be verified and color of its text is changed to red
    - When “done” is received it means that the current item has been checked and successfully verified if the item is automatically verifiable and it changes background to green

Web page also can also simulate user voice input and system automatic verification input:

- Key press “w”: Simulates “Start Power Up” user voice command
- Key press “s”: Simulates “Start Engine Failure” user voice command
- Key press “q”: Simulates automatic system verification of the current item
- Key press “e”: Simulates “Check” user voice command