**Internship Report - Isabel Soares**

1. Solutions found to generate a chatbot:
   1. **ChatterBot**

**DOCUMENTATION:**

To install, follow these steps:[Installation — ChatterBot 1.0.8 documentation](https://chatterbot.readthedocs.io/en/stable/setup.html)

To start, I found this tutorial that helped me very much in understanding better how ChatterBot works:[How to Create a Chatbot Using Chatterbot Python - YouTube](https://www.youtube.com/watch?v=ibu1Pjb6qEM)

**Limitations:**

* a simple FAQs; (*I found this limitation while implementing*)
* we cannot filter the information through the question; (*I found this limitation while implementing*)
* we cannot have more than 1 answer per question. (*I found this limitation while implementing*)

**Advantages:**

* Easy to use: chatbot answers questions similar to the Q&A.json(that was created by me) ([gunthercox/ChatterBot: ChatterBot is a machine learning, conversational dialog engine for creating chat bots (github.com)](https://github.com/gunthercox/ChatterBot) && [ChatterBot Tutorial — ChatterBot 1.0.8 documentation](https://chatterbot.readthedocs.io/en/stable/tutorial.html) ) and easy to add the data to the bot. That’s why this type of chat bot is good for FAQs, for instance.
* Have Corpus for many languages ([gunthercox/chatterbot-corpus: A multilingual dialog corpus (github.com)](https://github.com/gunthercox/chatterbot-corpus)).
* Use Search and Classifications Algorithms ([Frequently Asked Questions — ChatterBot 1.0.8 documentation](https://chatterbot.readthedocs.io/en/stable/faq.html#what-kinds-of-machine-learning-does-chatterbot-use)) which helps in compensating for users’ typos:

Basically, chat bot selects a response include:

[Search Algorithm]

* + the similarity of an input statement to known statements
  + the frequency in which similar known responses occur
  + the likeliness of an input statement to fit into a category that known statements are a part of.

[Classification Algorithm]

* Use naive Bayesian classification algorithms to determine if an input statement meets a particular set of criteria that warrant a response to be generated from that logic adapter.

**SIMPLE CONVERSATION EXAMPLE**

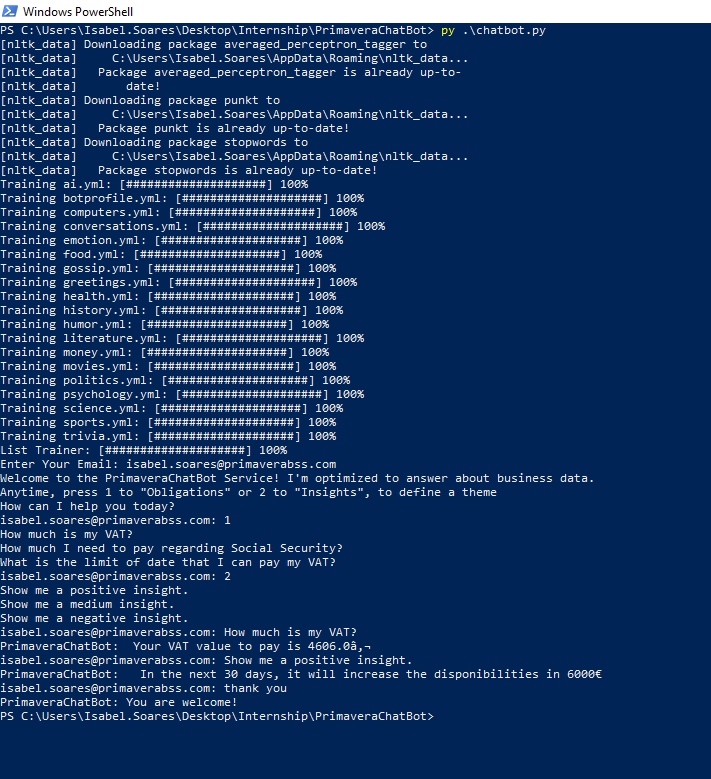
****

Figure 1 - A conversation using Chatterbot.

* 1. **Rasa**

**DOCUMENTATION:**

**Download**: [Installing Rasa Open Source: Windows 10 (Updated 2021) - YouTube](https://www.youtube.com/watch?v=GlR60CvTh8A)

Quick Installation - [Installation (rasa.com)](https://rasa.com/docs/rasa/installation)

Note: I needed to use administrator permissions, because Anaconda did not work without it.

I started to understand the documentation but, in my opinion, it is more useful to watch these videos from the Rasa YouTube Channel, although they show an old version (the main changes between the versions are in the format used, which was previously on Markdown and now is on YAML and the last version is easier and simpler than first, regarding to the policies and pipeline):<https://www.youtube.com/watch?v=-F6h43DRpcU&list=RDCMUCJ0V6493mLvqdiVwOKWBODQ&start_radio=1&t=12s>

In summary, Rasa divides itself in three majors’ components:

1. Rasa NLU - It is responsible for understanding the input messages of the user. It should recognize what the user wants and also extract information like names, places from the messages.
   1. Domain: defines the universe in which your bot operates. It specifies exactly:
      1. - which intents you are expecting to respond to
      2. - which slots you wish to track
      3. - which actions your bot can take
   2. Actions: are the things your bot can do.
   3. Stories: is a training data sample for the dialogue system. This shows the bot how to act and react to the inputs given by the user.
   4. Slots: are the things you want to keep track of during a conversation. The slots are the bot’s memory.
2. Rasa Core - It maintains the conversation flow between the User and the Bot.
3. Rasa X - is a toolset for developers to build, improve, and deploy contextual assistants with the Rasa framework. The good thing about Rasa X is that you can share your assistant with real users and collect the conversations they have with the bot, allowing you to improve it without interrupting the bot running in production.

Good summarize in : [How to build a contextual assistant using RASA Forms | by Amal Menzli | Towards Data Science](https://towardsdatascience.com/how-to-build-a-contextual-assistant-using-rasa-forms-82e014a10dea) ( I found very useful to understand, in a general way, the documentation of Rasa)

When you are running/training your bot use an Anaconda Powershell Prompt and the **list of commands** that you can use are listed in [Command Line Interface (rasa.com)](https://rasa.com/docs/rasa/command-line-interface)

A good way to understand how querying a database with a chatbot works is by watching this livecoding: [Rasa Livecoding: Querying a database with a chatbot - YouTube](https://www.youtube.com/watch?v=iyfJ0jx87w0) and the code is here (also very useful to understand how Rasa works with this example): [GitHub - rctatman/personal\_website\_bot: A simple bot to answer questions on my personal website. (In development)](https://github.com/rctatman/personal_website_bot)

**Limitations/Disadvantages:**

* more difficult in terms of understanding all the documentation; ([Introduction to Rasa Open Source](https://rasa.com/docs/rasa/))
* more difficult in terms of understanding how to connect our data to the bot, because it creates a database from our data (in my case .csv) (*I found this limitation while implementing and the links above helped me to overcome this limitation*)
* this is a limitation of rasa but I do not know the reason. I will give a tip that always helped me: when something is not working and you do not understand the bug, the best solution is to delete the models and the database of the obligations and insights. After, run again *rasa train* (to have a new a model) and create again new databases running the Jupiter notebook.
* By exploring some more functionalities of Rasa, I concluded that sometimes if the keyword was wrongly written, the bot will not understand and you can see it here:

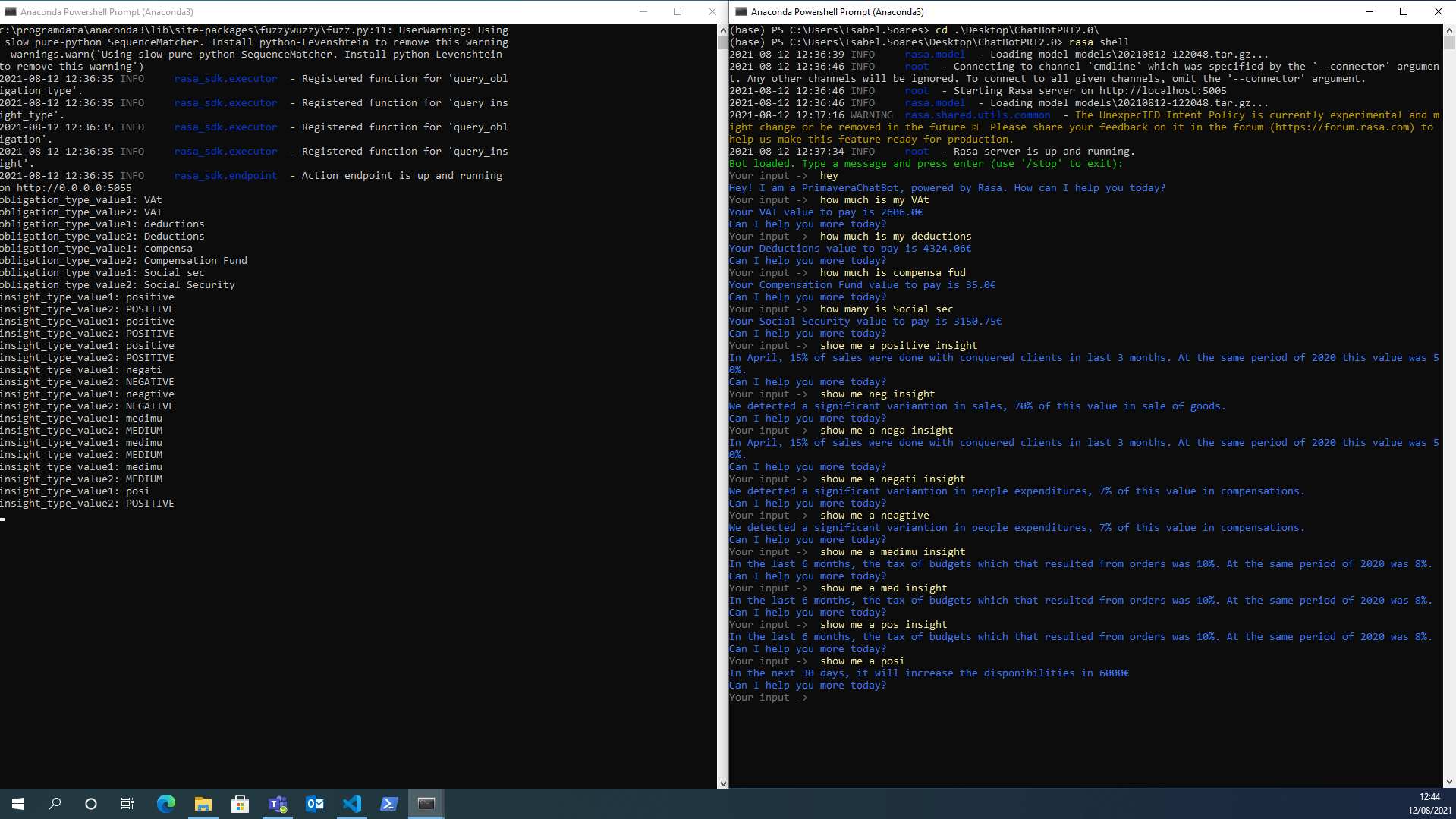


Figure 2 - A limitation of Rasa.

(NOTE: At the left Anaconda Powershell Prompt is where the actions associated to the database run and I added some prints to test what and if the bot understood the keyword even if it was not correctly written and at the right, you can see the conversation between bot and user)

**Advantages:**

* more powerful and complex tool than chatterbot, because the code of rasa keeps being updated and chatterbot is no longer updated (*I found this limitation comparing the documentation of both tools*);
* it also has a similar algorithm to ChatterBot, which tries to compensate for users’ typos.
* All the limitations of Chatterbot are solved with Rasa.

Both with Chatterbot and Rasa, I used some **type questions** regarding the *obligations*, such as:

* How much is my VAT?
* How much I need to pay regarding VAT?
* How much is my Social Security?
* How much I need to pay regarding Social Security?
* How much is my Deductions?
* How much I need to pay regarding Deductions?
* How much is my Compensation Fund?
* How much I need to pay regarding Compensation Fund?
* Until when can I pay my VAT?
* What is the limit of date that I can pay my VAT?
* Until when can I pay my Social Security?
* What is the limit of date that I can pay my Social Security?
* Until when can I pay my Deductions?
* What is the limit of date that I can pay my Deductions?
* Until when can I pay my Compensation Fund?
* What is the limit of date that I can pay my Compensation Fund?

And related to the *insights*, I used these:

* Show me a positive insight.
* Show me a medium insight.
* Show me a negative insight.
* Show me all insights.
* Show me all positive insights.
* Show me all medium insights.
* Show me all negative insights.

The PrimaveraChatBot is able to answer all these questions (the user can write the date on different format from the one presented below):

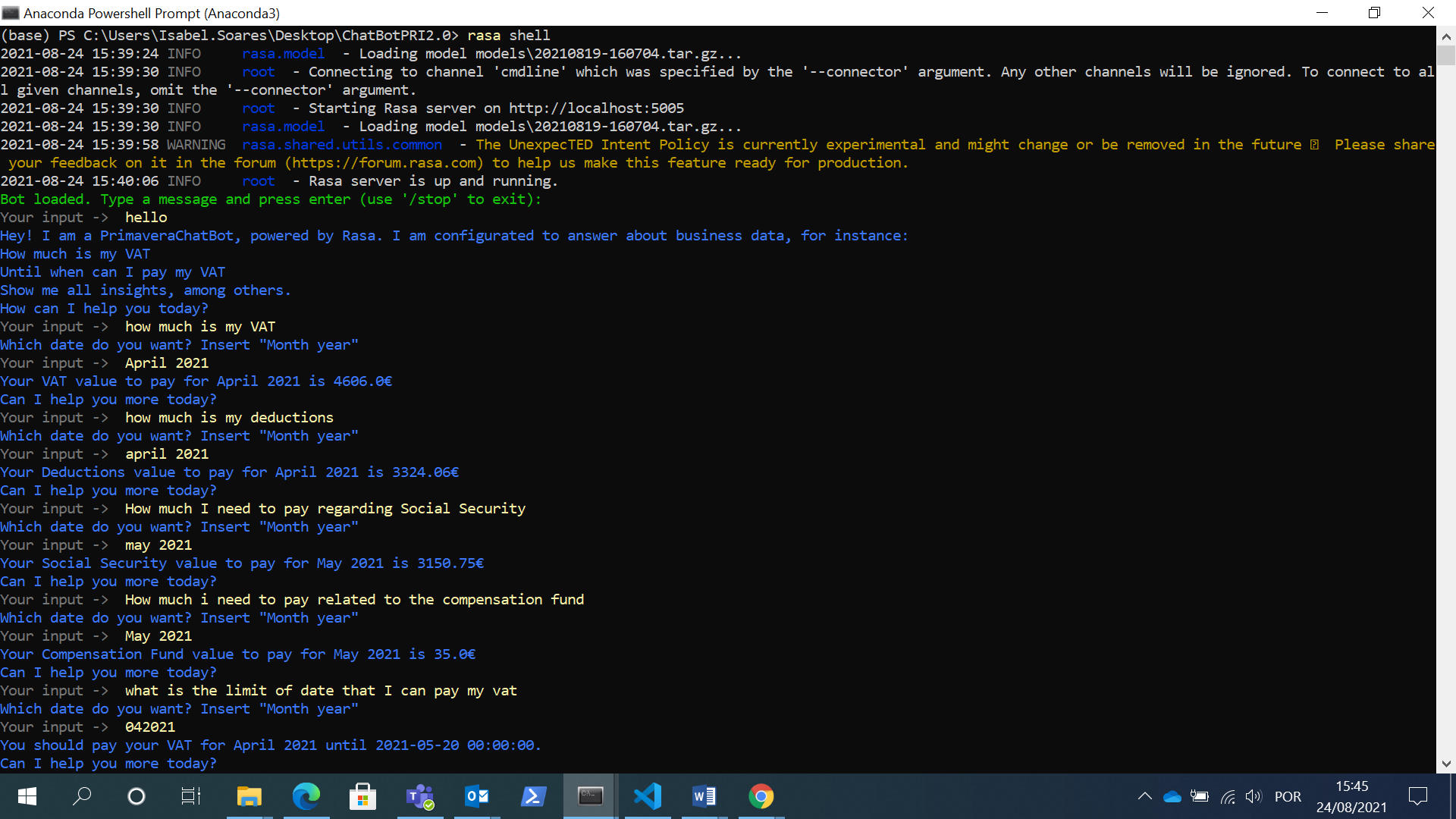


Figure 3 - The bot answers questions about the obligations Part I.

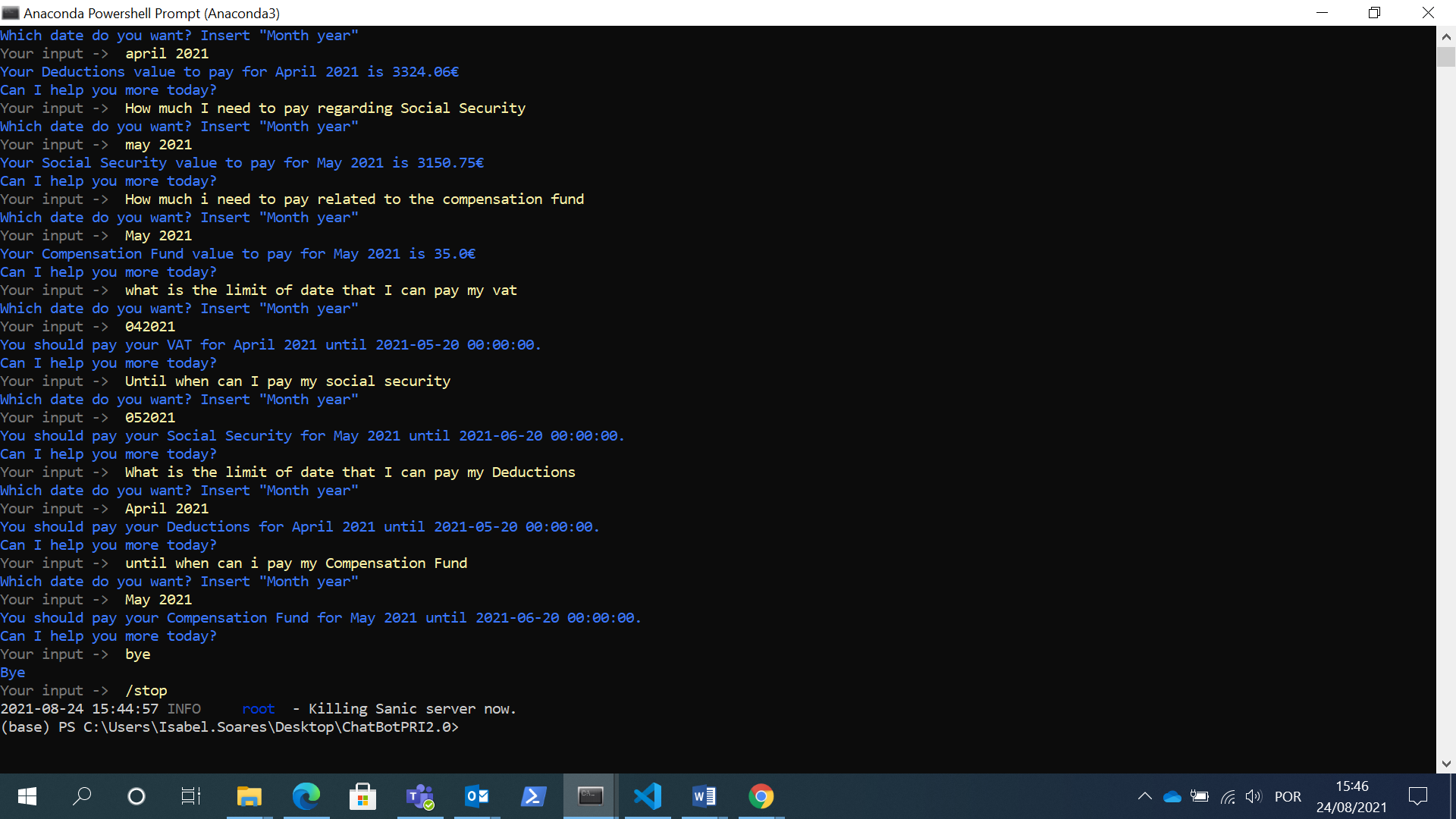


Figure 4 - The bot answers questions about the obligations Part II.

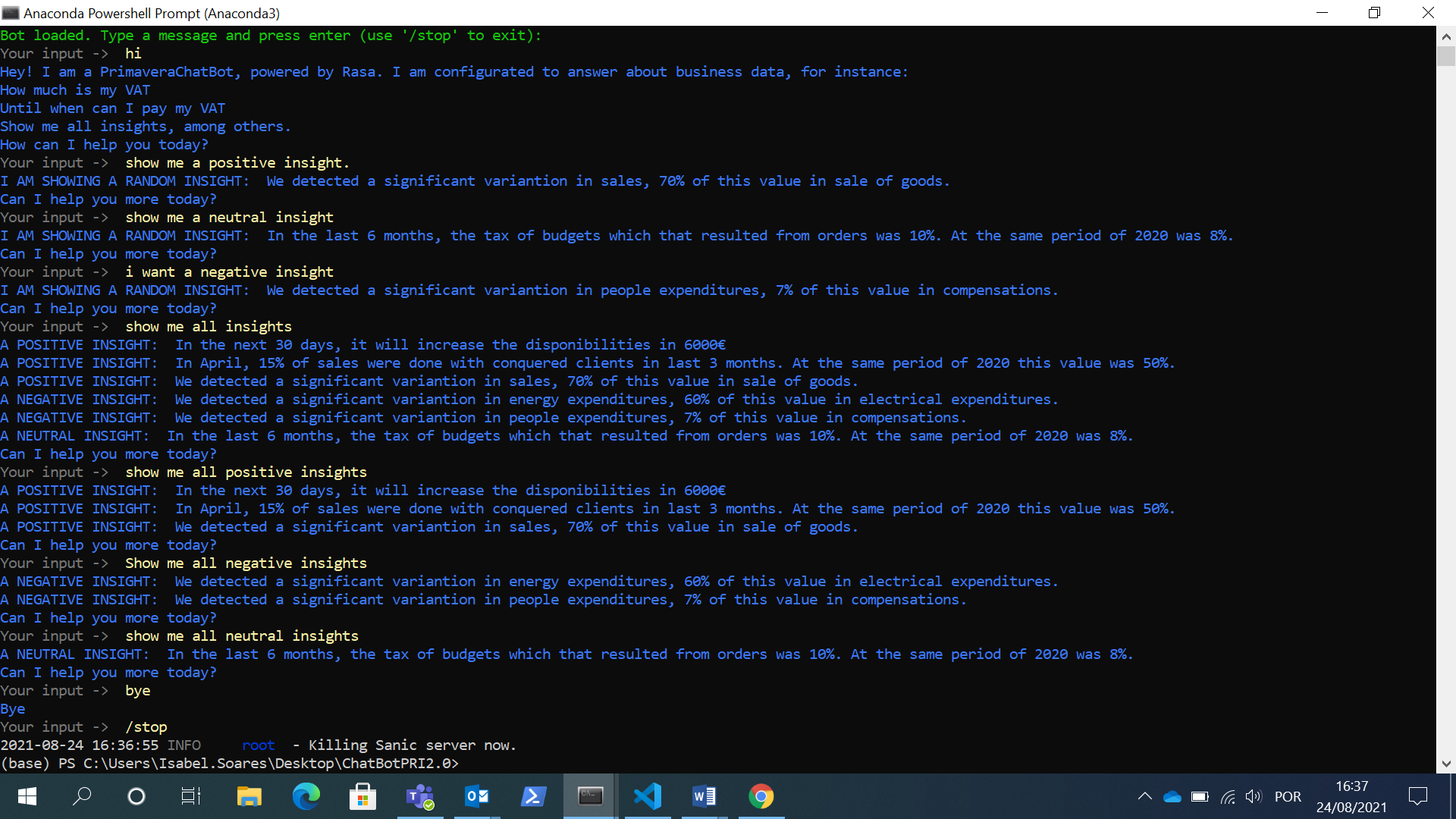


Figure 5 - The bot answers questions about the insights.

When I suppose that it can be a training problem or I want to understand what the bot is “thinking”, I used a rasa command, *rasa interactive*, which also helped me to see the confidence of each action and each intent. (The prints show the second time that I run it with this command)

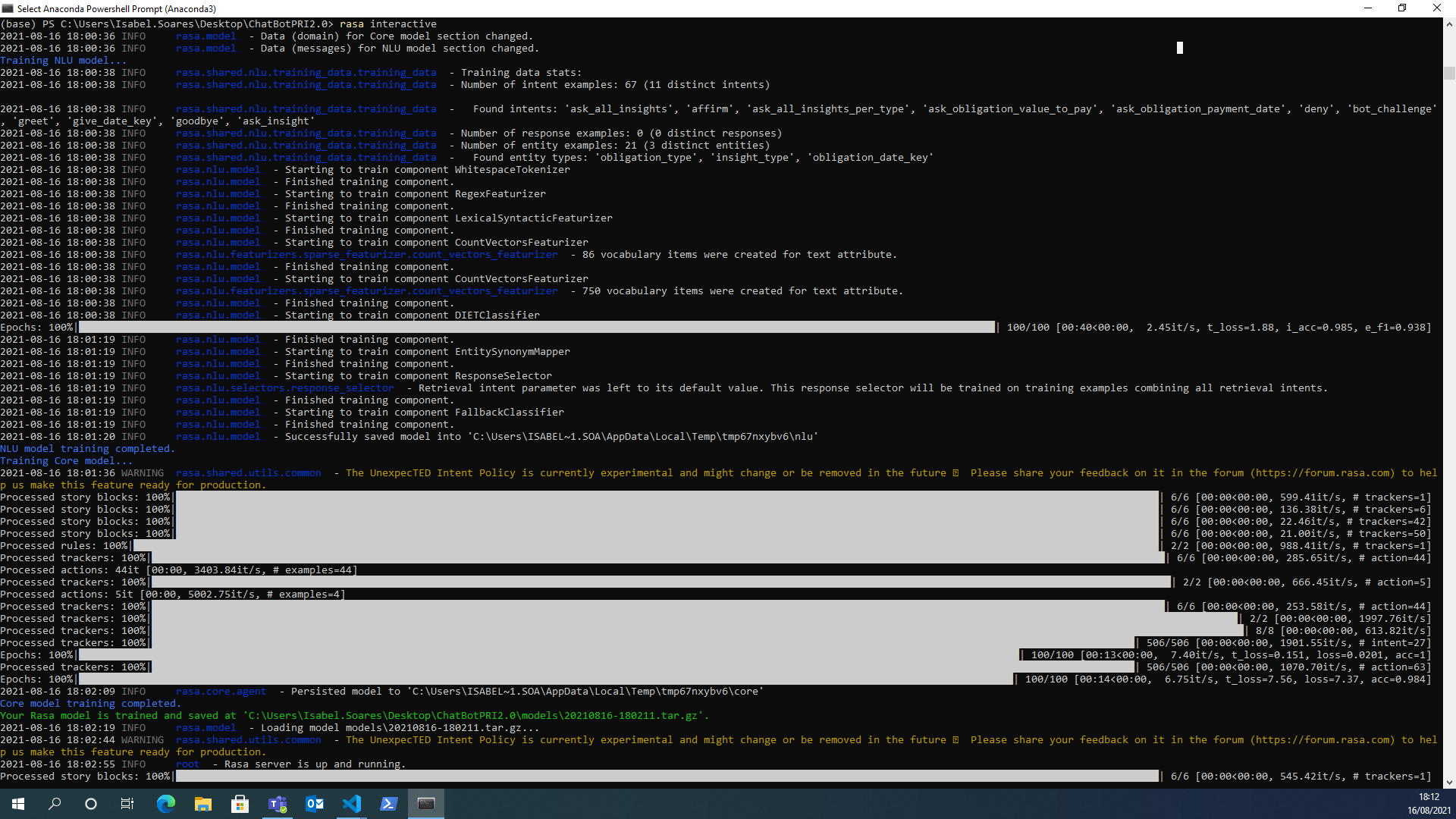


Figure 6 - Start running the command rasa interactive.

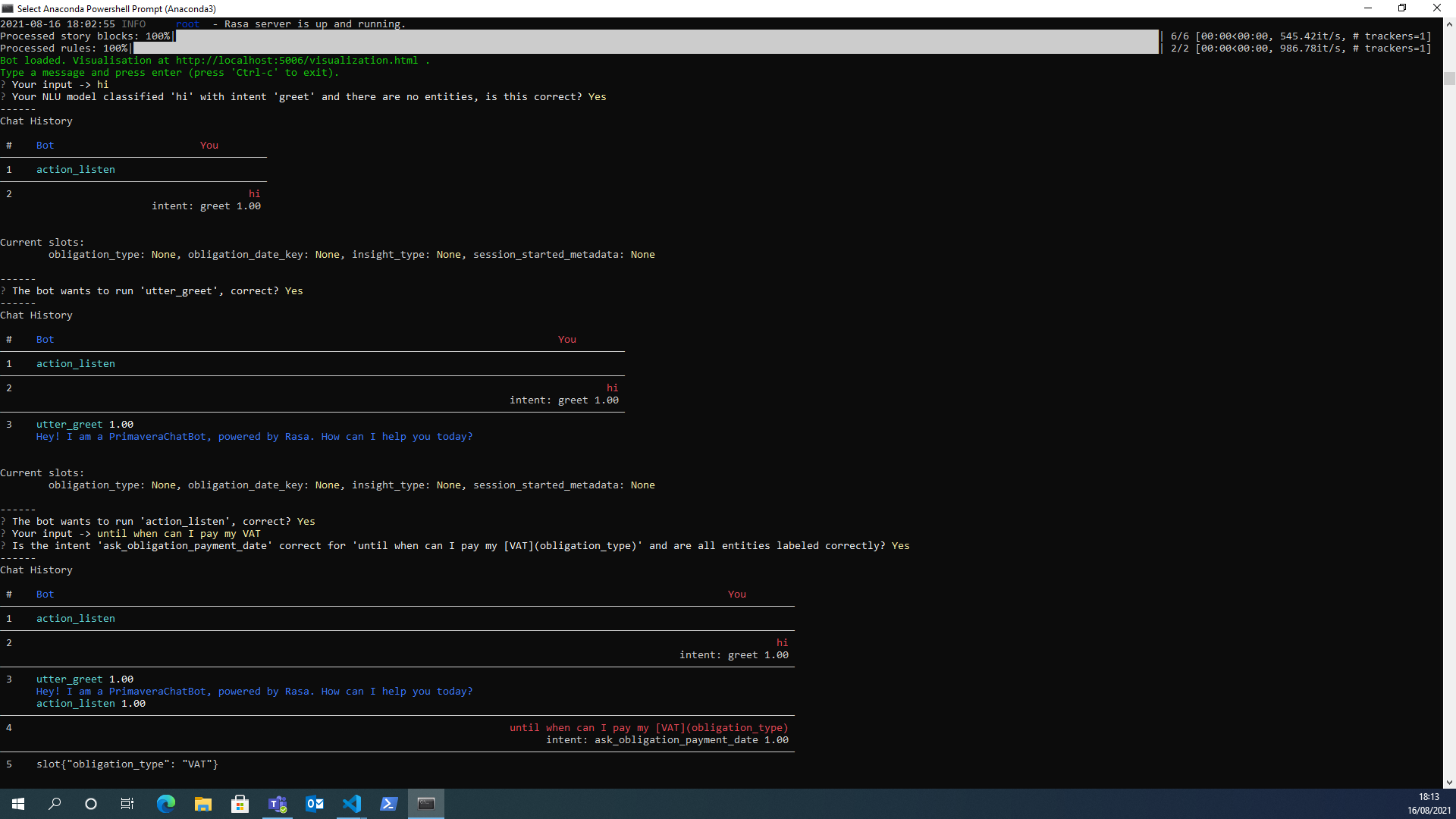


Figure 7 - Running rasa interactive (continuation).

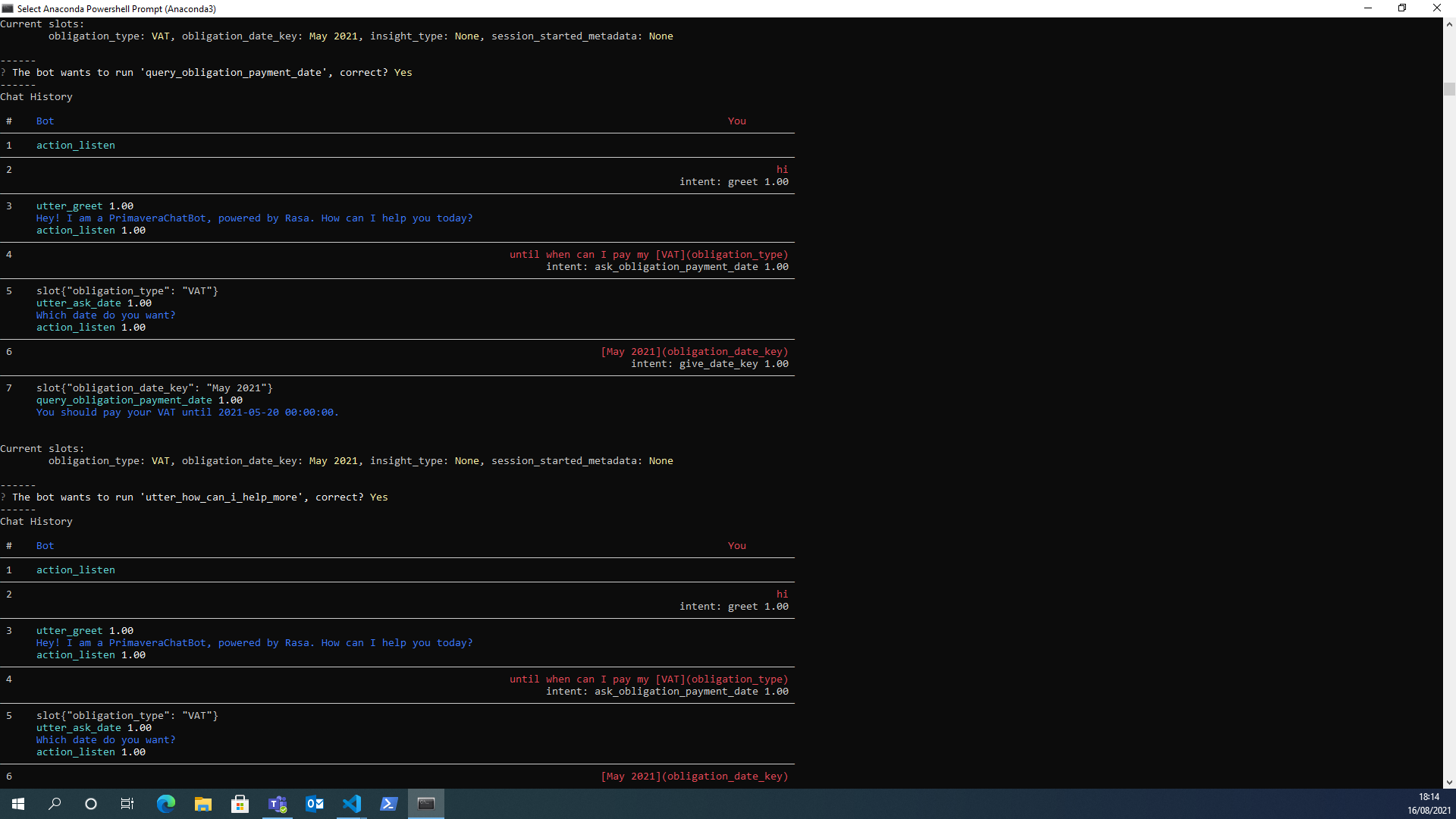


Figure 8 - Running rasa interactive (continuation).

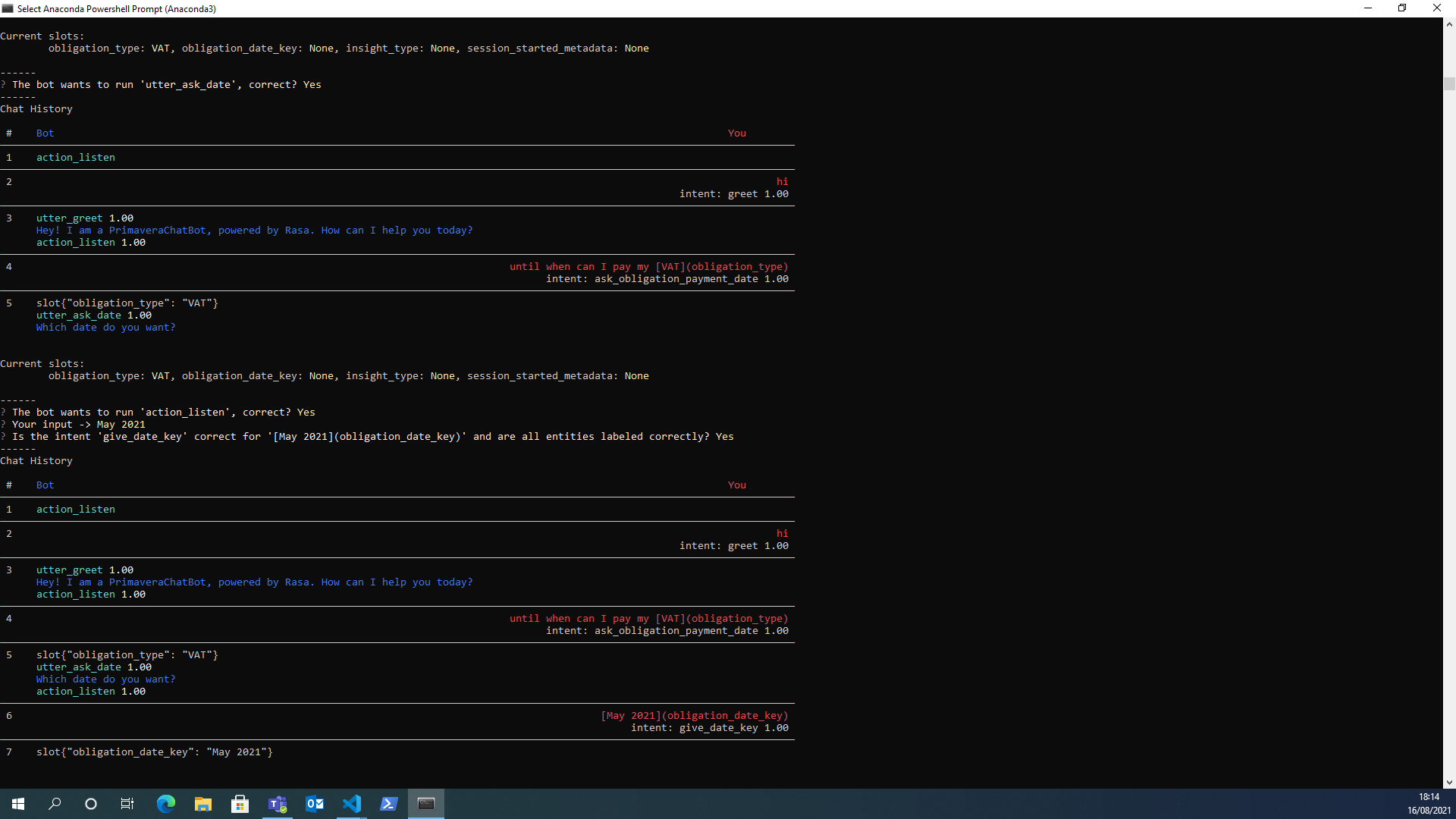


Figure 9 - Running rasa interactive (continuation).

Note that the value that appears next to the actions and intents is the confidence that the bot has in each of them.

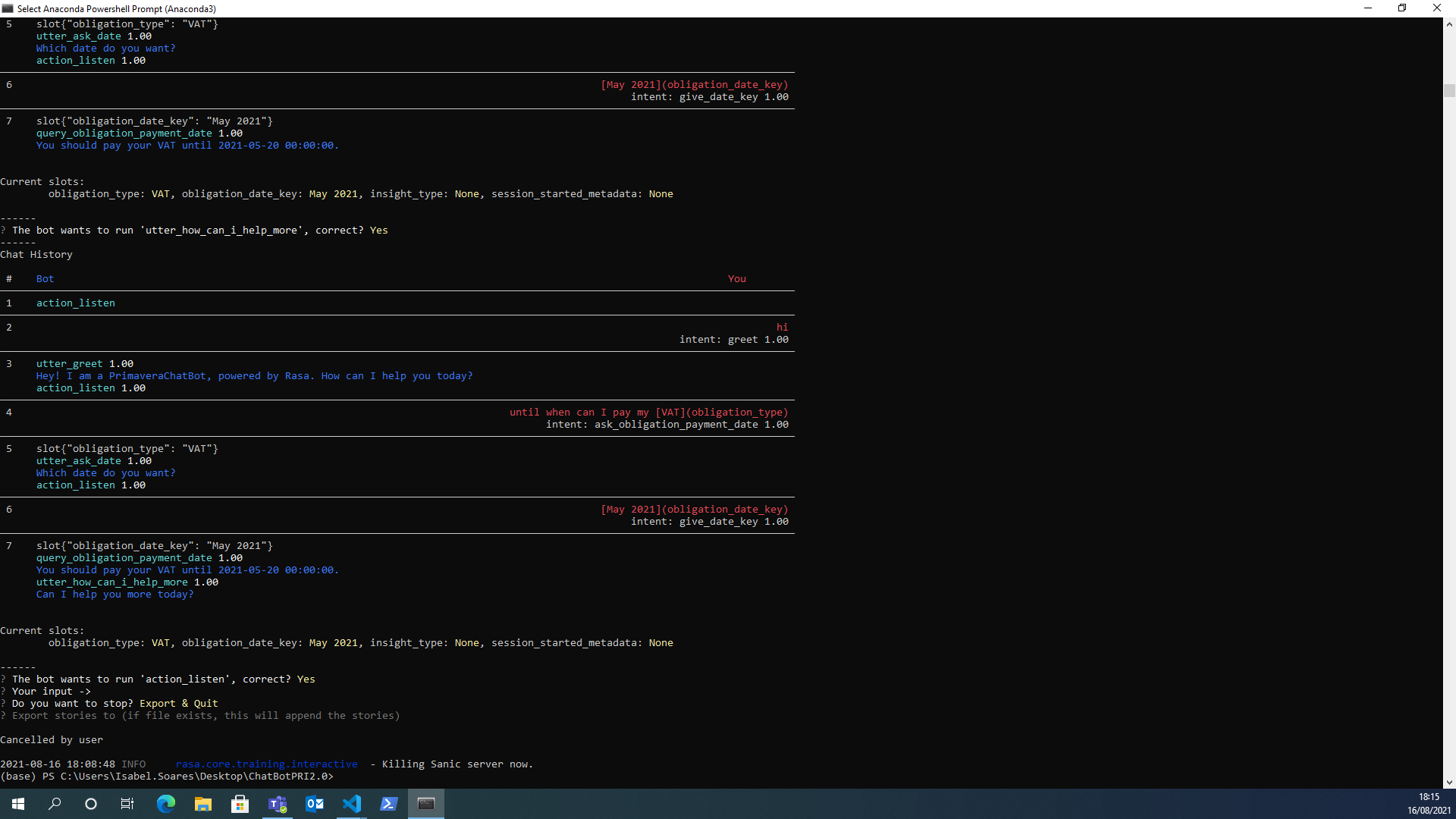


Figure 10 - Running rasa interactive (continuation).

While running *rasa interactive*, I could see a visualization of what was happening in real time with the bot:

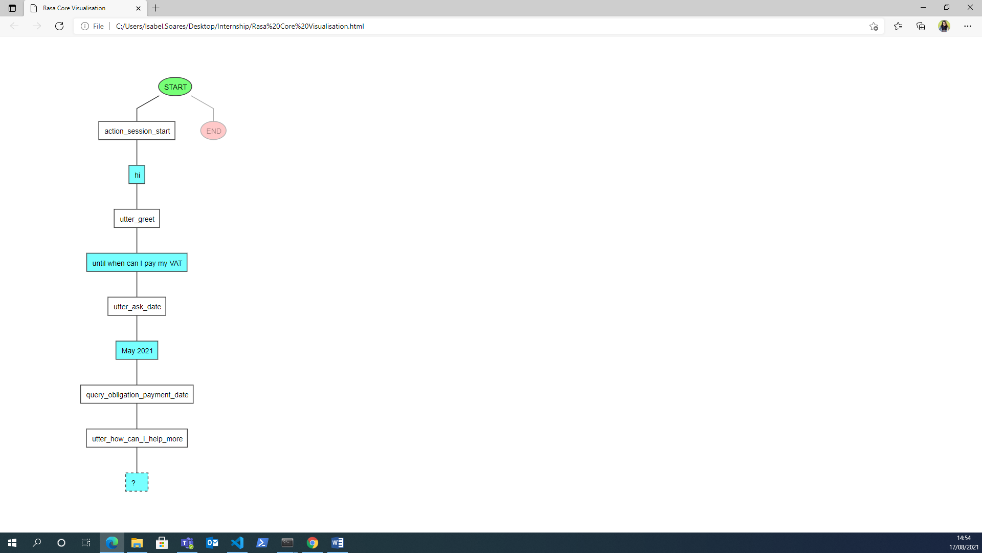


Figure 11 - Rasa visualization of the story.

Note: If you run the command *rasa visualize,* it generates a visual representation of your stories, like the one above.

1. Architectures
   1. **ChatterBot**

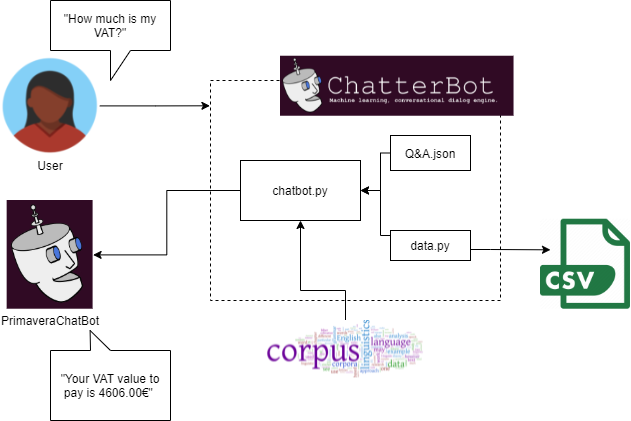
****

Figure 12 - ChatterBot Architecture.

* 1. **Rasa**

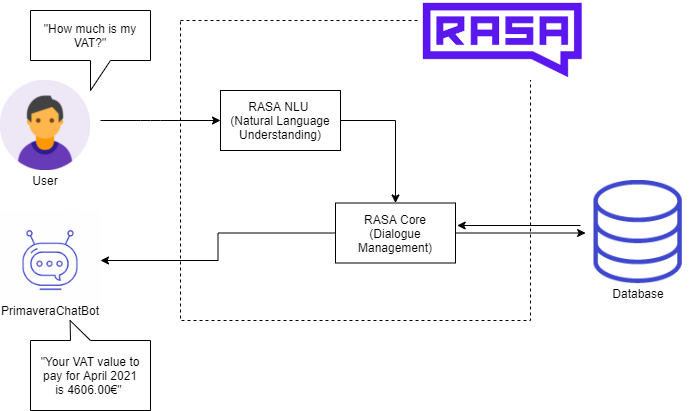
****

Figure - Rasa Architecture.

1. **Future Work**

* On the question “How much is my VAT?” (for instance), the bot should understand that it is implicit that it is regarding the last month;
* The possibility of adding date key to the question, for example “How much is my VAT for April 2021?”;
* If the user inserts some date or some information that there is not on the database, the bot answers with an error message;
* The bot gives a suggestion of the questioning path for the user to follow. For instance, the user presses 1 to ask questions about obligations and 2 to ask questions about insights and then, the bot only gets data about that certain topic;
* Train more the bot in order for the bot to always understand the keywords despite sometimes being wrongly written;
* Add voice to the chatbot (this link can be useful if you do not have installation problems like me: <https://rasa.com/blog/how-to-build-a-voice-assistant-with-open-source-rasa-and-mozilla-tools/>)