# Khởi tạo project

* Câu lệnh: rasa init --no-prompt (--no-prompt là bỏ qua qua các hỏi)
* Cấu trúc thư mục:
  + \_\_init\_\_.py: file empty giúp python tìm đc actions
  + actions.py: file chứa code action
  + config.yml: configure NLU và Core model
  + credentials.yml: file để kết nối các dịch vụ khác
  + data/nlu.md: dữ liệu để train NLU
  + data/stories.md: file stories
  + domain.yml: domain của assistant
  + endpoints.yml: thông tin để kết nối đến channel như fb messenger
  + models/<timestamp>.tar.gz: model ban đầu

# NLU training data

* Data Formats: Có thể cung cấp training data dưới dạng markdown hoặc JSON, dước dạng 1 file hoặc thư mục chứa nhiều file. (Markdown dễ sử dụng hơn)
* Markdown: là RASA NLU format dễ nhất cho con người đọc và viết. Có 2 cách viết:
  + [<entity text>](<entity name>)

VD: Could I pay in [yen](currency)?

* + [<entity-text>]{"entity": "<entity name>", "role": "<role name>", "group": "<group name>", "value": "<entity synonym>"}

VD: how much do I have on my [savings account]{"entity": "source\_account", "value": "savings"}

* Chú thích:
  + Entity: tên thực thể
  + Role: vai trò của thực thể đó
  + Group:
  + Value: từ đồng nghĩa với thực thể đó
* Tranning data cho RASA NLU đc chia thành các phần
  + Common example
  + Synonyms
  + Regex features and
  + Lookup table
* Tuy chỉ Common example là bắt buộc nhưng thêm các thành phần khác có thể tăng tính chắc chắn của NLU model
* Synonums sẽ map thực thể được xác định thành thực thể có tên cho trước (VD: “saving accout” thành “saving”). Tuy nhiên việc này chỉ có thể thực hiện đc khi đã xác định đc thực thể thành công
* Lookup table có tác dụng để tạo ra case-insensitive regex patterns để thêm vào regex features
* Regex feature sẽ có tác dụng trong trường hợp thực thể của bạn có cấu trúc xác định (VD: zipcode hoặc email address)
* JSON format:

{

**"rasa\_nlu\_data"**: {

**"common\_examples"**: [],

**"regex\_features"** : [],

**"lookup\_tables"** : [],

**"entity\_synonyms"**: []

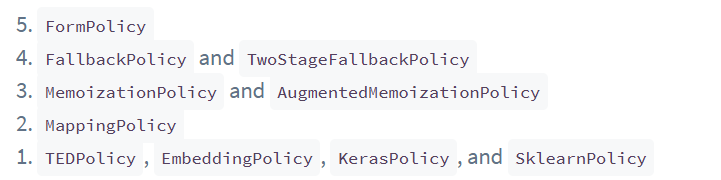
}

}

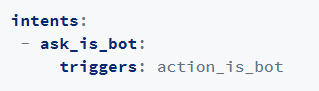
# Chọn Pipeline:

Trong RASA, các tin nhắn tới được sử lý bởi chuỗi các thành phần. Các thành phần này đc thực hiện lần lượt trong 1 process đc gọi là pipeline được định nghĩa trong config.yml

* Policies:
  + Max history: số lượng cuộc hội thoại model dựa vào để đưa ra quyết định hiện tại
  + Data Augmentation: Khi bạn train model, Rasa Core sẽ mặc định tạo ra những stories dài hơn với những stories trong file của bạn. Nếu bạn muốn bỏ qua mặc định này và chỉ muốn bot trả lời ngay câu hỏi hiện tại dựa theo stories của bạn định sẵn thì sử dụng –augmentation flag
  + Action Selection: Mỗi policy được định nghĩa trong file config sẽ dự đoán hành động tiếp theo với 1 sự chắc chắn nhất định. Hành động tiếp theo của bot sẽ được dựa vào policy có sự chắc chắn cao nhất. Trong TH 2 policy có sự chắc chắn bằng nhau thì sẽ dựa theo rank của chúng

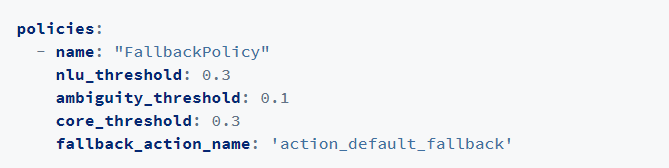


* + Keras Policy sử dụng mạng nơ-ron được định nghĩa trong Keras để dự đoán hành động tiếp theo
  + TED Policy
    - epochs: số lần thuật toán chạy với dữ liệu training. 1 epochs tương ứng với 1 forward pass và backward pass của tất cả dữ liệu training
    - hidden\_layers\_sizes
    - number\_of\_transformer\_layers
    - transformer\_size
    - weight\_sparsity
  + Mapping Policy: dùng để map intent với 1 action

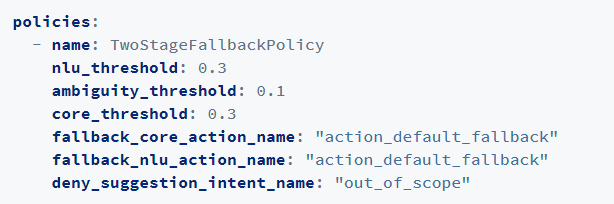


1 intent có thể được gán với tối đa 1 action

* + Memoization Policy: ghi nhớ những cuộc trò chuyện trong dữ liệu training. Nếu một cuộc trò chuyện chính xác với dữ liệu training thì dự đoán với tính chắc chắn 1.0 ngược lại sẽ dự đoán 0.0
  + Augmented Memoization Policy: giống như Memoization Policy nhưng chỉ nhớ trong giới hạn max\_history
  + Fallback Policy: kích hoạt fallback action (VD :”Xin lỗi tôi không hiểu bạn đang nói gì”) khi một trong những trường hợp sau xảy ra:
    - Sự chắc chắn trong nhận dạng intent thấp hơn nlu\_threshold
    - Sự chắc chắn trong intent chắc chắn nhất với intent chắc chắn thứ 2 ít hơn ambiguity\_threshold
    - Không một policy nào có sự chắc chắn cao hơn core\_threshold



* + Two-Stage Fallback Policy: dùng để sử lý sự chắc chắn thấp trong NLU
    - Nếu NLU prediction thấp hoặc không lớn hơn sự chắc chắn intent chắc chắn thứ 2 thì người dùng sẽ được hỏi để xác nhận intent của mình và nếu người dùng xác nhận đúng thì tiếp tục và nếu không thì yêu cầu nhập lại tin nhắn
    - Trường hợp nhập nếu intent của người dùng được đoán đúng thì sẽ tiếp tục. Nếu intent được đoán với sự chắc chắn thấp thì sẽ yêu cầu người dùng xác nhận intent của mình
    - Nếu người dùng xác nhận intent là đúng thì tiếp tục. Ngược lại intent là sai thì sẽ gọi tới action được khai báo ở deny\_suggestion\_intent\_name



* + Form Policy: một dạng mở rộng của Memoization Policy dùng trong form

# Stories:

Rasa stories là 1 dạng training data được sử dụng để train model quản lý hội thoại của Rasa. Một story là 1 ví dụ của 1 cuộc trò chuyện giữa người dùng và trợ lý AI được biến đổi thành 1 format cố định. Trong đó dữ liệu đầu vào của người dùng được hiểu là intent (và entities nếu cần thiết) và response của trợ lý là action

## Cấu trúc của một story:

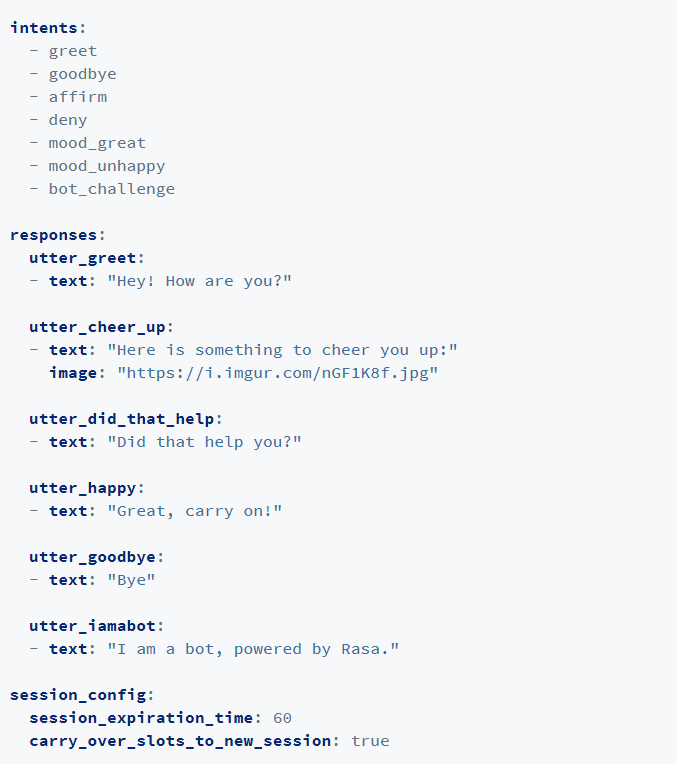
* + Tên story bắt đầu bằng ## và có thể đặt tên tùy thích (VD: ## story\_03248462 )
  + Tin nhắn được gửi bởi người dùng bắt đầu bằng \* và có dạng intent{"entity1": "value", "entity2": "value"}
  + Action được bot thực hiện bắt đầu bằng –
  + Event được trả lại bởi 1 action nằm cùng 1 dòng ngay sau action đó

# Domain:

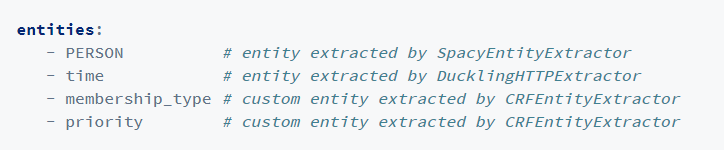
Domain định nghĩa giới hạn của trợ lý của bạn bao gồm: những dữ liệu đầu vào của người dùng, hành động nên thực hiện, dữ lieu nên lưu trữ là gì.

Trong file domain chứa intents, entities, slots, và actions của bot của bạn.

Xong khi chạy xong câu lệnh rasa init, file domain.yml sinh ra đã được định nghĩa sẵn:



Nếu chúng ta cần xác định entities trong câu nói của người dùng thì cần liệt kể entities có thể được xác định bởi NLU pipeline



Slots chứa những thông tin mà bạn muốn theo dõi trong quá trình trò chuyện

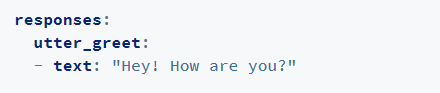


Actions là những thứ con bot của bạn có thể là được. VD như là: trả lời người dùng, thực hiện gọi đến một API, query DB, …

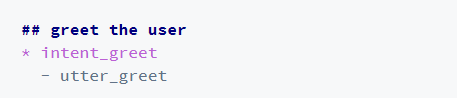
Để khai báo custom action trong file domain chúng ta khai báo cái tên mà action đó trả về

Để viết response của bot cho người dùng có 2 cách viết.

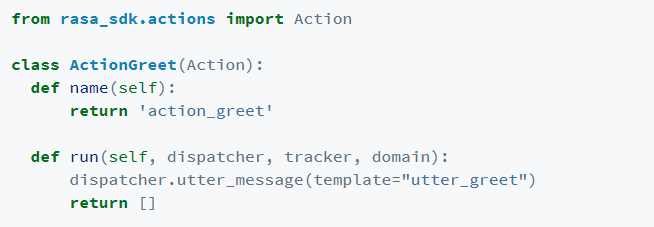
* Nếu tên response bắt đằu bằng utter\_ thì có thể được sử dụng trực tiếp như một action. Thêm response vào file domain



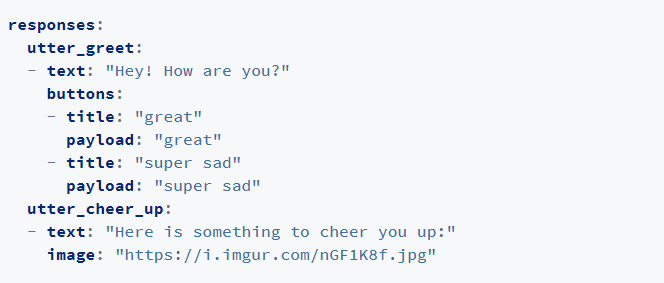
Sau đó có thể sử dụng response đó như một action trong file stories



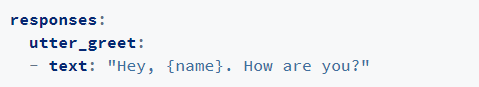
* Ngoài ra có thể tự tạo response message từ custom action bằng cách sử dụng dispatcher để gửi tin nhắn cho người dùng



Có thể thêm ảnh hoặc button vào trong response



Có thể truyền biến vào trong response. VD bạn có utter response như sau

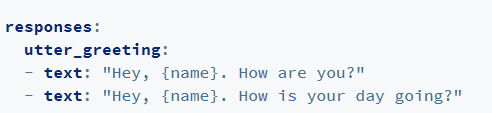


Rasa sẽ tự động điền vào biến name với giá trị tìm được trong slots được gọi là name

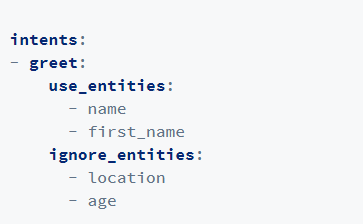
Ngoài ra có thể viết custom action như sau



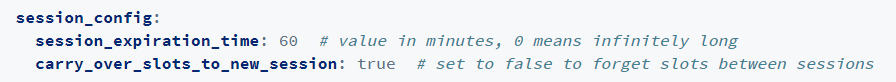
Một utter response có thể có nhiều response message và Rasa sẽ chọn random 1 response message để gửi lại cho người dùng



Có thể khai báo những entity nào được sử dụng trong intent nào bằng cách thêm use\_entities và ignore\_entities



Có thể cài đặt thời gian session tồn tại trong Rasa bằng cách khai báo session\_config trong file domain



* session\_expiration\_time là thời gian không hoạt động cho phép của session
* carry\_over\_slots\_to\_new\_session là setting để quyết định xem có đưa slot trong session cũ sang session mới hay không

# Train model:

Mỗi khi chúng ta thêm dữ liệu NLU hoặc dữ liệu Core mới hoặc cập nhật file domain, chúng ta cần re-train mạng nơ-ron cho stories và dữ liệu NLU. Chúng ta có thể sử dụng câu lệnh rasa train