**ACTOR & USE CASE RECONIZER MODEL DOCUMENTATION VERSION 1.6**

**Overview**

Model ini digunakan untuk mengidentifikasi actor dan use case dalam sebuah paragraph dalam konteks use case specification

**Prerequisite**

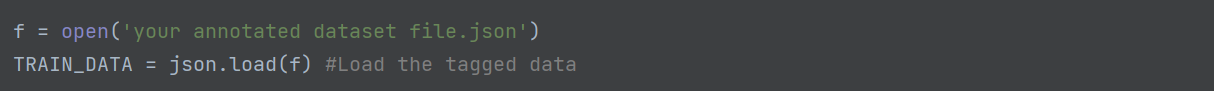
* Phyton 3.10.11
* Python packages :

1. Spacy
2. Json
3. Tqdm
4. Inflect
5. Transformers

**Model Algorithm**

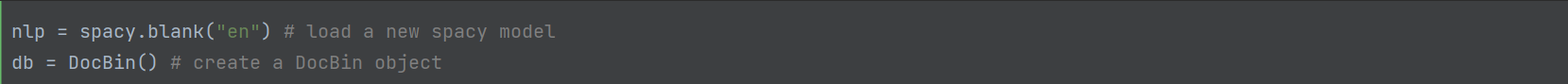
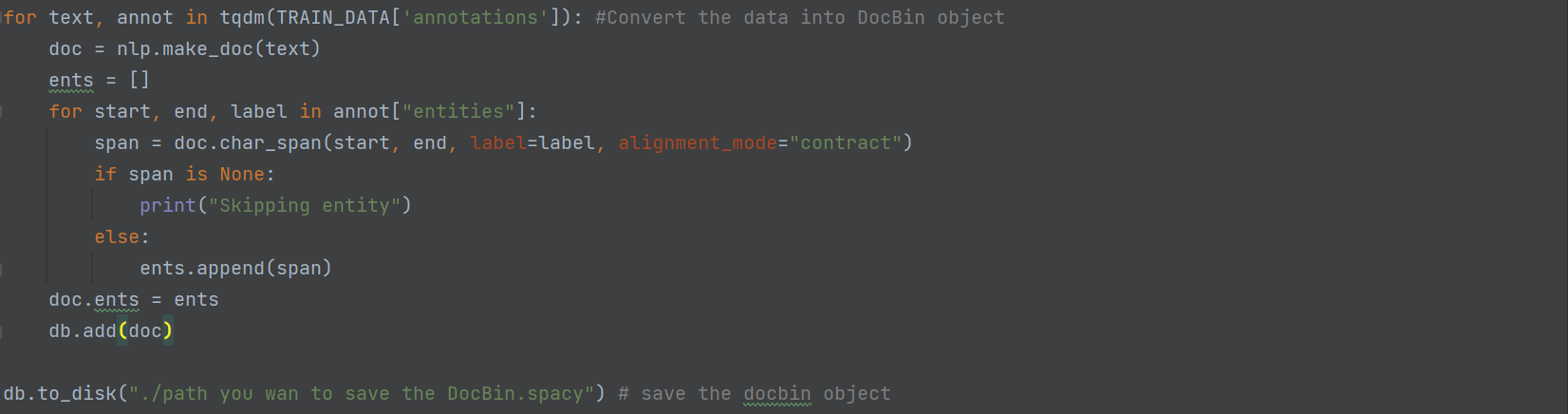
1. Load annotated dataset

Load annotated dataset in form of json file. You can annotate text dataset from [NER Annotator for SpaCy - https://tecoholic.github.io/ner-annotator/](https://tecoholic.github.io/ner-annotator/) then download the file as json type file. Then import it using json library like code below :

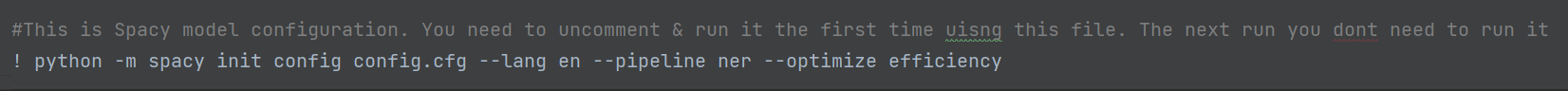


1. Convert annotated dataset into SpaCy DocBin object

In spaCy, a DocBin object is a container for storing and efficiently serializing batches of Doc objects. A Doc object in spaCy represents a processed text document. Before converting dataset into DocBin object you need to initialize Spacy model and DocBin object first like code below :

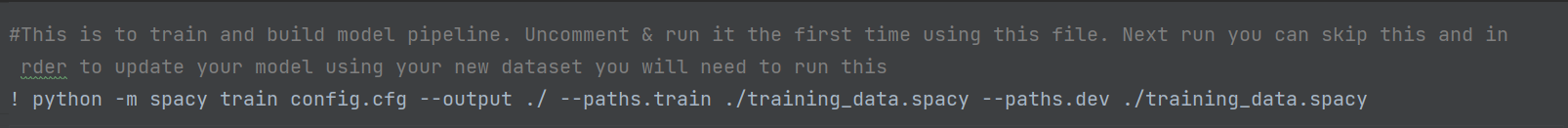
  
Then saved the DocBin object into a file.spacy into a folder path you want to choose like code below :   


1. Train SpaCy NER model

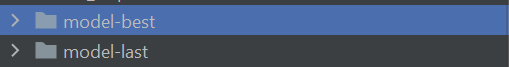
First create a configuration file to train the model. To Create the configuration you can visit Spacy documentation to choose the configuration that suit your model at [Doc · spaCy API Documentation - https://spacy.io/api/doc/](https://spacy.io/api/doc/) In this model configuration use English language, for ner task pipeline and low resource configuration to optimize efficiency like code below :  


Then it will add new file in the directory project named config.cfg

Then after train the model using the configuration that already set at config.cfg file like the code below :

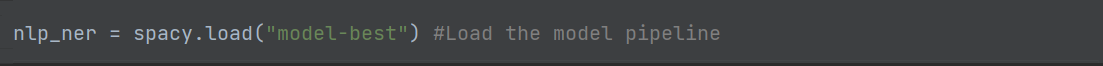


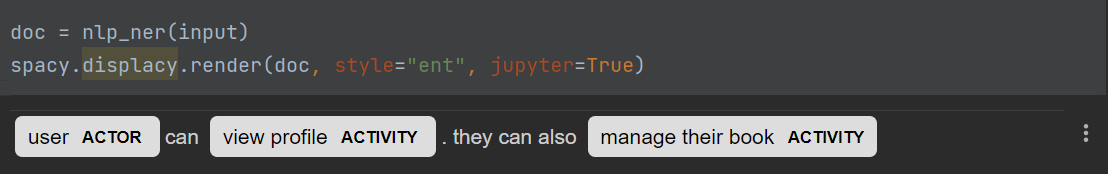
The code above will train model and then create a new directory for model. Change the ./training\_data.spacy into your docbin file name that already generated in step 2. After run the code it will generate 2 new directory named model-best and model-last. In spaCy, when you train a model, it generates multiple snapshots or checkpoints during the training process. Each snapshot represents the state of the model at a specific point in training. These snapshots are saved in the model's output directory, and they are typically named with an incrementing number. The "best" model refers to the snapshot that performed the best on the evaluation data during training. The best model is often determined based on metrics like precision, recall, F1 score, or other relevant evaluation metrics for the ner. The "last" model refers to the snapshot taken at the end of the training process, regardless of its performance on the evaluation data. In this model model-best will be used



1. Load the trained model

To load the model you can load using code bellow using spacy library :

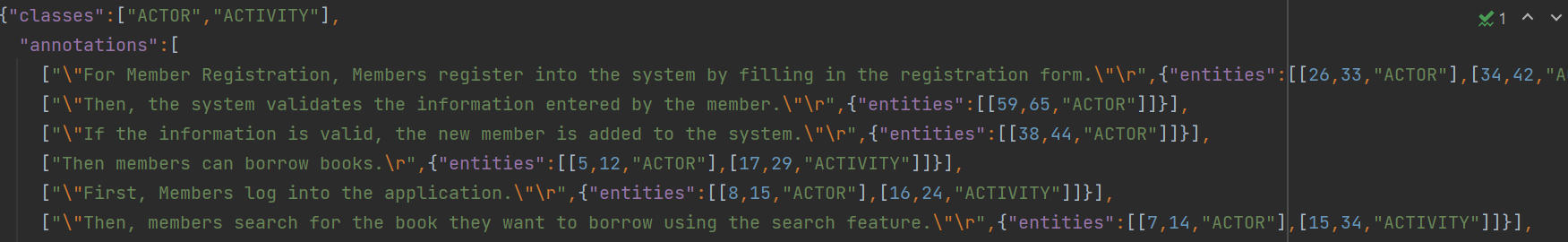


The model will recognize which word is a actor or use case from a paragraph like visualization bellow : 

**Dataset**

The dataset used in the model consist of 11 paragraph or 124 sentences that has language style for people for people with domain knowledge of use case. The text data is example of input that will be used in the model. Each paragraph in dataset describe how the requirements of the system will be created in point of view of developer. Here a example of dataset language style :

For Member Registration, Members register into the system by filling in the registration form. Then, the system validates the information entered by the member. If the information is valid, the new member is added to the system.

The dataset then annotated using text annotator from [NER Annotator for SpaCy - https://tecoholic.github.io/ner-annotator/](https://tecoholic.github.io/ner-annotator/) an then converted into json file. There are two entities into each annotation which is actor and use case. Here an example of it :  


**Implementation of Model :**The model implemented into a file named actor\_model.py In this file consist a function to process the paragraph into a dictionary output named actors\_dict with actors as keys and the use case as its values. Here the code of the function.

import spacy  
import inflect  
  
model = spacy.load('model-best')  
  
def recognize\_actor(input):  
 nlp\_sents = spacy.load("en\_core\_web\_sm")  
 nlp\_sents.add\_pipe("sentencizer") # Add sentence boundary detection  
  
 p = inflect.engine()  
 actors = set()  
 usecase = set()  
 actors\_dict = {}  
 current\_actor = None  
 input = input.lower()  
 doc = model(input)  
  
 for ent in doc.ents:  
 if ent.label\_ == "ACTOR":  
 singular\_actor = p.singular\_noun(ent.text) or ent.text  
 actors.add(singular\_actor)  
 elif ent.label\_ == "ACTIVITY":  
 usecase.add(ent.text)  
  
 for sent in nlp\_sents(input).sents:  
 for ent in model(sent.text).ents:  
 if ent.label\_ == "ACTOR":  
 current\_actor = p.singular\_noun(ent.text) or ent.text  
 elif ent.label\_ == "ACTIVITY":  
 actor\_key = current\_actor or actors\_dict.get(None)  
 if actor\_key not in actors\_dict:  
 actors\_dict[actor\_key] = set()  
 actors\_dict[actor\_key].add(ent.text)  
  
 return actors\_dict

The function only take one string argument as its input and it process each sentence per sentence. First input will transform into lower case. The inputs then processed by the model to recognize which is actor and which use case. After that each actors and use case assign into set list variable to remove duplicate. In the actor list, there may be contains actor with single or plural word which is actually one actor. To prevent that before assigned into a set list variable (actors), first the actor will be process with inflect engine. This engine will change all plural actor into a single actor.

Because the model process sentence per sentence in a paragraph, then it will be a problem when determine the association of use case and the actor when the user input the sentences without an actor in a sentence. To prevent that the function will be assign the use case without an actor mention before in a sentence into a last actor that processed. This will be assumed the use case is associated with the actor in the last sentence. Then actor will assigned into dictionary as a key and its use case as its value into variable name actors\_dict.

**WhiteBox Unit Testing Documentation on Function**

This documentation is testing the function above for some combination of input that user possibly input in the function. The automated unit testing saved into a file named unittest\_actor.py. Here are some test case of it :

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Input | Output | Is it Acceptable |
| Simple sentence | '''User can view profile''' | 'user': {'view profile'} | Yes |
| Simple two sentences with pronouns | '''User can view profile. They can also manage their book''' | 'user': {'view profile', 'manage their book book'} | Yes |
| 2 sentences. first sentence with actor second without actor | '''User can view profile. They can also manage their book''' | 'user': {'view profile', 'manage their book book'} | Yes |
| 1 sentence with exception | '''User can view profile but user cannot view other profile''' | 'user': {'view profile'} | Yes |
| 2 actor in 1 sentence | User and administrator can view profile | {'administrator': {'view profile'}} | No |
| 1 sentence, 2 actor with exception | '''User can view profile but user cannot view other profile''' | 'user': {'view profile'} | Yes |
| Passive phrases | '''employee only can be hired by manager''' | {} | No |
|  |  |  |  |