Wit.ai - Movie bot

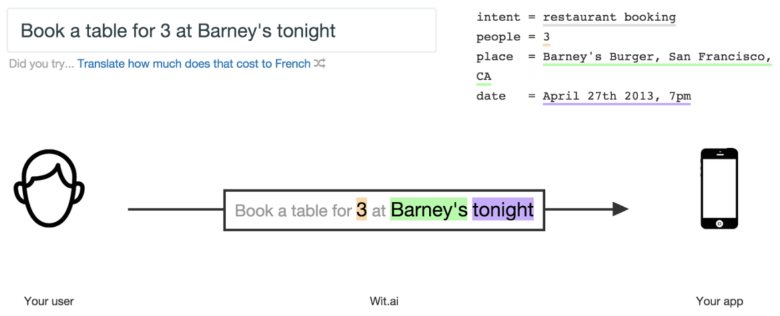
Exercises

What is the MovieBot? Generally we want to design a chatbot that knows everything about the movies. This chatbot should understand and reply back to the following questions:

- tell me about the untouchable
- who directed the untouchable?
- when was the untouchable released?

According to Figure 1, the user phrase should be translated to the chatbot app. As we saw in the previous lessons, intents, and other related name entities should be extracted from each phrase. As an example in this figure, a 'restaurant booking' intent should extract three more name entities: people, place and date.

In order to extract intents and entities, we introduced the Regular expressions. To be able to use the Regexs, several cases should be defined manually


Book a table for 3 at Barney's tonight
Did you try... Translate how much does that cost to French
Book a table for 3 at Barney's tonight
Your user
Wit.ai
Your app

```
intent = restaurant booking
people = 3
place  = Barney's Burger, San Francisco,
CA
date   = April 27th 2013, 7pm
```

Figure 1: Turn what users say in action

which is not evident in every situation. For this reason, we are interested in Machine learning (ML), specifically in Natural Language Processing (NLP).

$$\boxed{\text{what is the weather in Istanbul?}} \longrightarrow \boxed{\begin{array}{c}\text{Machine learning}\\ \text{Natural language}\\ \text{processing}\end{array}} \longrightarrow \text{what is the } \underbrace{weather}_{intent} \text{ in } \underbrace{Istanbul}_{place}?$$

To use the NLP methods you don't need too many information for implementing intelligent approaches at the moment. Different approaches will be introduced in the future. There exist several NLP services such as: Watson's Conversation Service (WCS), Microsoft's Language Understanding Intelligence Service(LUIS), Google Natural Language(NL) API, wit.ai and Dialogue flow.

Exe 1 **start with the movieBot** Take your prepared code from the previous weeks and continue this project on your code. If you still struggle with the code, download the prepared project in the Brightspace and continue with the rest.

reminder If you haven't done it, just be sure to accomplish the steps for creating a facebook app for your chatbot, get the page access token, verify token and etc and modify them in the 'config' folder. Setup an HTTPS for your webhooks as well. For more information check the subjects of week3 and week4.

Check your package.json file and coherency of its parameters with your package versions and files names inside the project file.

Exe 2 Run this code on your facebook page by sending a message to your chatbot and check if it works properly. If you use the supported code in the brightsapce you will have what the chatbot.

Exe 3 In the facebook developer dashboard related to your chatbot, there is anothe option namely 'Built-in NLP'. We are not going to use it in our project, but it is a provided option for simple natural language processing problems in the chatbot, based on Facebook messenger server `https://developers.facebook.com/docs/messenger-platform/built-in-nlp`.

Just for checking this option turn it on in your Facebook developer dashboard and check what is its output for the following examples after running your server.js and writing the following messages in your chatbot inside your facebook page.

- hello, how are you doing?
- what is the weather in Los Angeles?
- my phone number is 0235486554

To test these messages you should add a line in the server.js file:

```
console.log(message.nlp.entities)
```

Question It is up to you to find its place in the server.js file.

Since we are interested in using more intelligent methods, we are interested in other services, especially here in wit.ai.

Wit.ai is a natural language understanding and processing engine owned by Facebook that makes it easy for developers to build bots, applications and devices that we can talk to or text to. Wit.ai is a cloud service API for speech recognition and natural language processing, to use with bots, applications and devices.

Wit.ai can help with:

- Understand Natural Language: parse a message (Voice or Text) into structured data.
- Converse: predict the next action the bot should do (Bot Engine).

For extracting intents and entities, the wit.ai should be trained on various examples.

Notice For better understanding exercises 3 to 6, read this introduction precisely: `https://wit.ai/docs/quickstart`

Exe 3 Sign in to wit.ai (`https://wit.ai/`). create a new App for your moviebot namely moviebot. Watching this video will help you with doing this exercise: `https://www.youtube.com/watch?v=9DE4WV5w6zM`

Exe 4 Go to your App dashboard and select the understanding tab on the top right. This is where you can train your bot. Train your bot for understanding the following sentences and extracting their intents and entities as the following examples.

- tell me about the lord of the rings $\longrightarrow$ intent: movieinfo, movie: lord of the rings
- tell me about the spider man $\longrightarrow$ intent: movieinfo, movie: spider man
- tell me about the eternal sunshine of spotless mind $\longrightarrow$ intent: movieinfo, movie: eternal sunshine of spotless mind
- tell me about the spider man (2007) $\longrightarrow$ intent: movieinfo, movie: spider man, releaseYear: 2007
- tell me about the spider man (2012) $\longrightarrow$ intent: movieinfo, movie: spider man, releaseYear: 2012

To better training your model, you should give more similar examples to the system. This is based on artificial intelligent and machine learning methods. More you train your model, better it predicts the considered goals.

If you click on intents, you can see the list of expressions, while by clicking on free texts and keywords you can see the list of keywords. You can improve your system by defining synonyms for your keywords.

Exe 5 Train the wit.ai for director intent and movie according to the following examples:

- who directed the lord of the rings? $\longrightarrow$ intent: director, movie: the lord of the rings
- who directed the spider man? $\longrightarrow$ intent: director, movie: spider man
- tell me about how I met your mother.
- who directed lord of rings?

keep training the system with more examples.

Exe 6 Train the wit.ai for the releaseYear intent and movie according to the following examples:

– when was the untouchable released? $\longrightarrow$ intent: releaseYear, movie: untouchable

– when was the interstellar released? $\longrightarrow$ intent: releaseYear, movie: interstellar

Exe 7 **integrate wit.ai in Built-in NLP** In order to use any supported NLP service in the facebook platform, you should integrate it in the FB app using the Built-in NLP option. Go to the setting of your wit.ai, you can see its App ID, server access token and other information. The 'service access token' should be copied in the facebook app dashboard. in the Built-in NLP part of your setting in the facebook app, click the 'select a page' button, choose your facebook page assigned to the chatbot and paste the service access token[1] in the Wit Token section.

Question Now the NLP analyzer of your chatbot is wit.ai. Rerun your server.js file and see the different on the output results.

When the awi.ai sends us back the entities, we need to extract them first. After extracting entities (intent, movie, releasedYear), we need some functions for fetching some information of the asked movie from the 'themoviedb.org' as a movie database. Lets call these functions:

- getMovieData()
- getDirector()

Afterward, we should translate the fetched information from the movie database and send it back to the facebook messenger user.

Exe 8 **extractEntity** Make a folder namely 'tmdb' in your project and create an index.js file inside it. Write a function namely 'extractEntity' for receiving nlp data (gathered from the wit.ai) and entity name which returns back the value of the selected entity from the nlp data.

```
const extractEntity = (nlp, entity) =>{
  //should be filled by you.
}
```

[1]coming from the wit.ai

notice that the nlp data, received from the wit.ai is an object with the following format:

```
{
"intent":"movieinfo"
"entities":{
  "movie":"Spiderman"
  "releasedYear": "2002"
  }
  "confidence":0.83
}
```

write the 'extractEntity' function such that it returns back the entity value with a confidence equal or higher that 0.8, otherwise it returns back null.

Exe 9 For exporting this function to other files, you can export your module using the promise.

```
module.exports = nlpData => {
    return new Promise((resolve, reject) {
        let intent = extractEntity(nlpData, 'intent');
        resolve(intent);
    });
}
```

notice that this is a suggested approach, you implement the code as you prefer.

Notice Node.js: The promise constructor takes one argument, a callback with two parameters, resolve and reject. It then does something within the callback, perhaps async, then call resolve if everything worked, otherwise call reject. similar to throw in the JavaScript, it's customary, but it is not always required to reject with an Error object.

Question: Import the tmdb in your serve.js file. In the server.js (more precisely in server.post(), print the entity value extracted by your EextractEntity function. Now test it with various examples.

Exe 10 **moviDB api** After extracting name entities, we need a database (API) for fetching the desired information from. ThemovieDB is the one: `https://www.themoviedb.org/`. Create an account, connect to the page, go to the setting, then chose the API tab in the left side list. Copy the API key, and add it to your config/development.json file.

```
{
  "FB": {
    "PAGE_ACCESS_TOKEN": "page_access_token",
    "VERIFY_TOKEN": "verify_token",
    "APP_SECRET": "app_secret"
  },
  "TMDB": //API_key from themoviDB here
}
```

Now call this value in config/index.js. For the sake of your convinient the index.js file should be modified as below:

```
'use strict';

if(process.env.NODE_ENV === 'production') {
  module.exports = {
    FB: {
      PAGE_ACCESS_TOKEN: process.env.PAGE_ACCESS_TOKEN,
      VERIFY_TOKEN: process.env.VERIFY_TOKEN
    },
    TMDB: process.env.TMDB
    //this line has been added for calling the api key.
  }
} else {
  module.exports = require('./development.json');
}
```

Exe 11 **Quick look at MovieDB** To have a better understanding of the MovieDB and its returned query, go back to the movieDB dashboard and click on the 'developers.themoviedb.org'. You will be directed to a new page including some introductions and information in the get started in the left list bar. With respect to your moviebot, you can use various endpoints. For instance one interesting endpoint for the movie bot is SEARCH, for search movies. You can find it in the left bar list. After clicking on the search movies, you will find query string parameters. The interesting point is that, you can 'try it out' manually.

Go to the 'try it out', paste the same api key and write a movie name for the query cell. Click the send request button. Check the response, and check diferent format of the results as: pretty, JSON Explorer and raw. This shows you how to search the required details from the fetched query of the movieDB.

In your query, your results has an id. This id can be used for getting more information on your requested movie (Copy this id). For instance, select 'movies' from the left bar list and then select 'Get Credits' from the list. 'Try it out' manually. Paste the movie id, check the api key and finally send request. See the results, you have several arrays in the json out pout, cast, crews, etc. Find your movie director from the query result. Check the JSON Explorer as well here.

Notice Node js: **await**[2] The await operator is used to wait for a Promise. It can only be used inside an async function.

Syntax: `[rv] = await expression;`

rv: Returns the fulfilled value of the promise, or the value itself, if it's not a `Promise`.

The await expression causes async function execution to pause until a `Promise` is resolved, that is fulfilled or rejected, and to resume execution of the async function after fulfillment. When resumed, the value of the await expression is that of the fulfilled `Promise`. If the `Promise` is rejected, the await expression throws the rejected value. If the value of the expression following the await operator is not a `Promise`, it's converted to a resolved `Promise`.

Example If a `Promise` is passed to an `await` expression, it waits for the `Promise` to be fulfilled and returns the fulfilled value.

```
function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function f1() {
  var x = await resolveAfter2Seconds(10);
  console.log(x); // 10
}
f1();
```

If the value is not a Promise, it converts the value to a resolved Promise, and waits for it.

[2]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

```
async function f2() {
  var y = await 20;
  console.log(y); // 20
}
f2();
```

If the Promise is rejected, the rejected value is thrown.

```
async function f3() {
  try {
    var z = await Promise.reject(30);
  } catch(e) {
    console.log(e); // 30
  }
}
f3();
```

Handle rejected Promise without try block.

```

var response = await promisedFunction().catch((err) => {
    console.log(err); });
// response will be undefined if the promise is rejected
```

Exe 12 **getMovieData()** After querying themovieDB, we are interested in getting the following list of information from the query result: id, title, overview, release_date and whatever you like to extract such as the poster_path for getting the movie poster image. In tmdb/index.js, write a function namely 'getMovieData()' that gets two parameters, movie name and release year and returns back the query from theMovieDB.

```
const getMovieData = (movie, releaseYear = null) => {
  //filled by you
    return new Promise((resolve, reject) => {
      //filled by you.
    });
}
```

Use the request module in nodejs[3] for fetching the query from the movieDB api. Notice that the requires parameters for requesting a query to the movieDB api are including `api_key:TMDB`, movie name and release year. You also require a correct url adress for connecting

[3]`https://www.twilio.com/blog/2017/08/http-requests-in-node-js.html`

to the moviedb, search movie. As an example an url as `https://api.themoviedb.org/3/search/movie`. Try to use promise for writing this code.

Now returns back this output in your module.exports by using the Promise and await. This part of code is given completely.

```
module.exports = nlpData => {
    return new Promise(async function(resolve, reject) {
        let intent = extractEntity(nlpData, 'intent');

        if(intent) {
            let movie = extractEntity(nlpData, 'movie');
            let releaseYear = extractEntity(nlpData, '
                releaseYear');

            // Get data (including id) about the movie
            try {
                let movieData = await getMovieData(movie,
                    releaseYear);
                resolve(response);
            } catch(error) {
                reject(error);
            }

        } else {
            resolve({
                txt: "I'm not sure I understand you!"
            });
        }

    });
}
```

Test your code.

Notice node.js[4] Rewriting callback-based Node.js applications. Async functions return a Promise by default, so you can rewrite any callback based function to use Promises, then await their resolution.

Exe 13 **getDirector()** Go to tmdb/index.js and try to write the getDirector() function such that it receives the movieid and returns back the movie

[4]`https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function`

director by using the `Promise`, requesting a query to the MovieDB. You can export this value in your export module and getting the movie id by the getMovieData() function. Test your code.

Exe 14 After getting the required information from your MovieDB, modify your code in a way that responds back the required information to the user in the Facebook messenger. These responses are including the:

- movieinfo or
- director name

according to the requested question from the user in FB messenger.
To make your chatbot more interesting, you can get the movie poster(as an image) as well, and upload it back to the user in the FB messenger. This element is presented as the `poster_path` as a query result from the MovieDB.