ALEX - MovieBOT

Technical Design Document

Table of Contents

1. Introduction 3

1.1 Business Functionality 3

1.2 NLP using Recast.ai 4

1.3 Recast.ai Skills 8

1.4 NodeJS Layer 10

1.4.1 Packages 10

1.4.2 Routes 10

1.5 WebChat Integration 13

1.6 Deploy Recast.ai Bot 15

1.7 References 15

1. Introduction

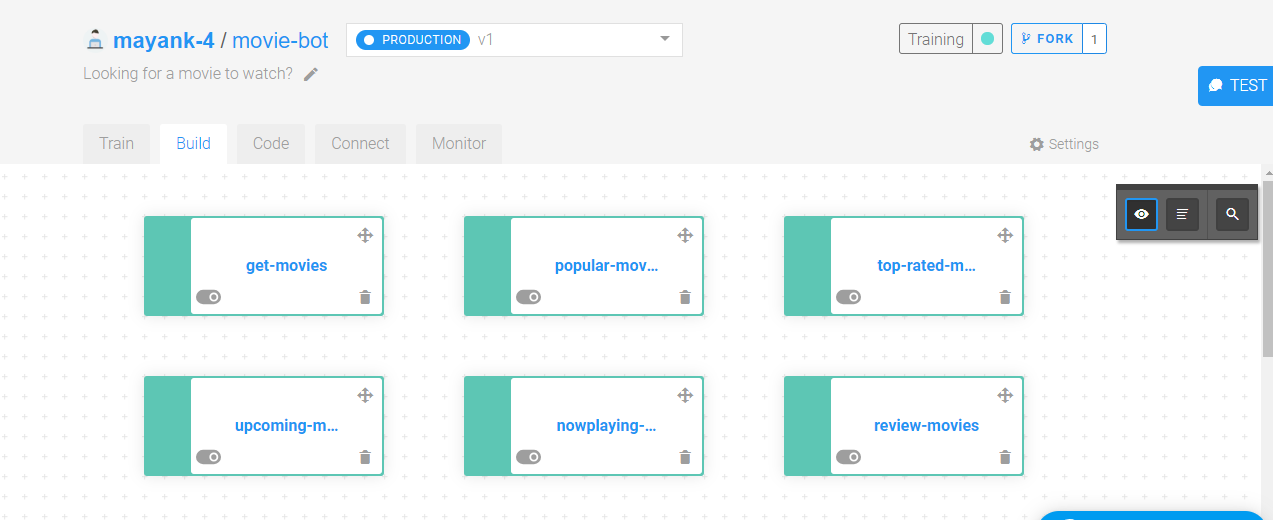
1.1 Business Functionality

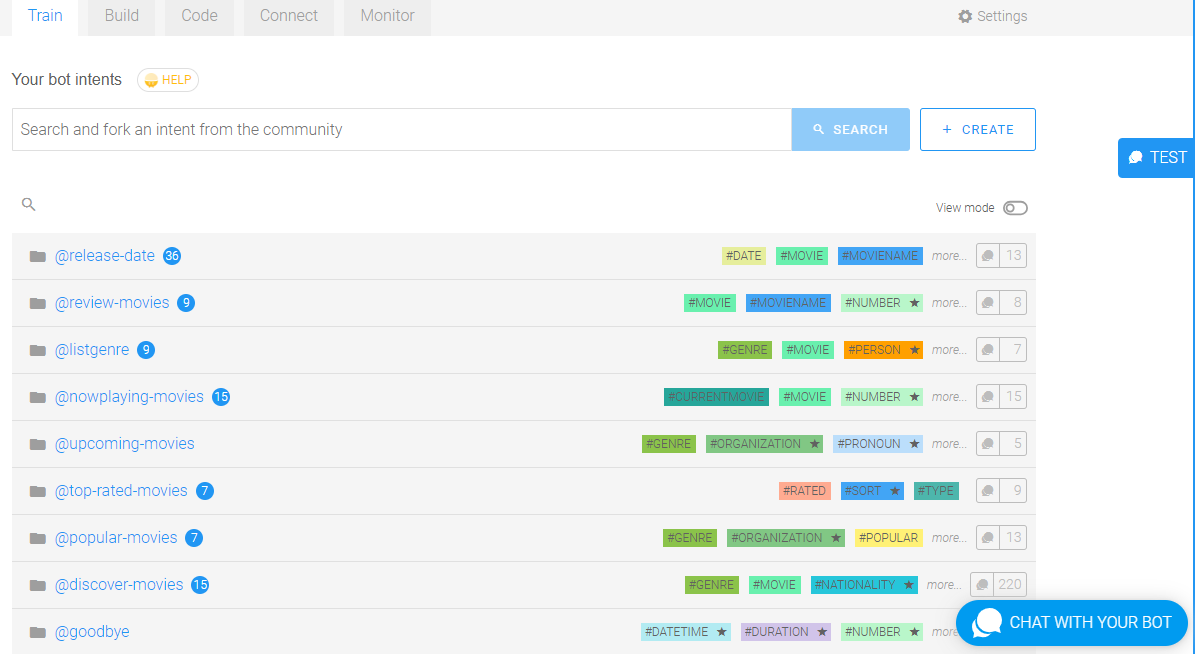
**ALEX** is an intelligent chatbot designed to improve the user experience of an end user in providing movie watching assistance. Its existing use cases are focused on going through different websites to find the desired results.

**ALEX** acts as a digital assistant which show movies on basis of - Popular, top rated, now playing, up-coming categories. ALEX also provides movies recommendation based on genre of a movies such as action, drama, comedy, romance etc. and can help assisting using reviews of movie and release date of the movie. This brings about a new level of efficiency and simplicity for the end user to use the movie bot.

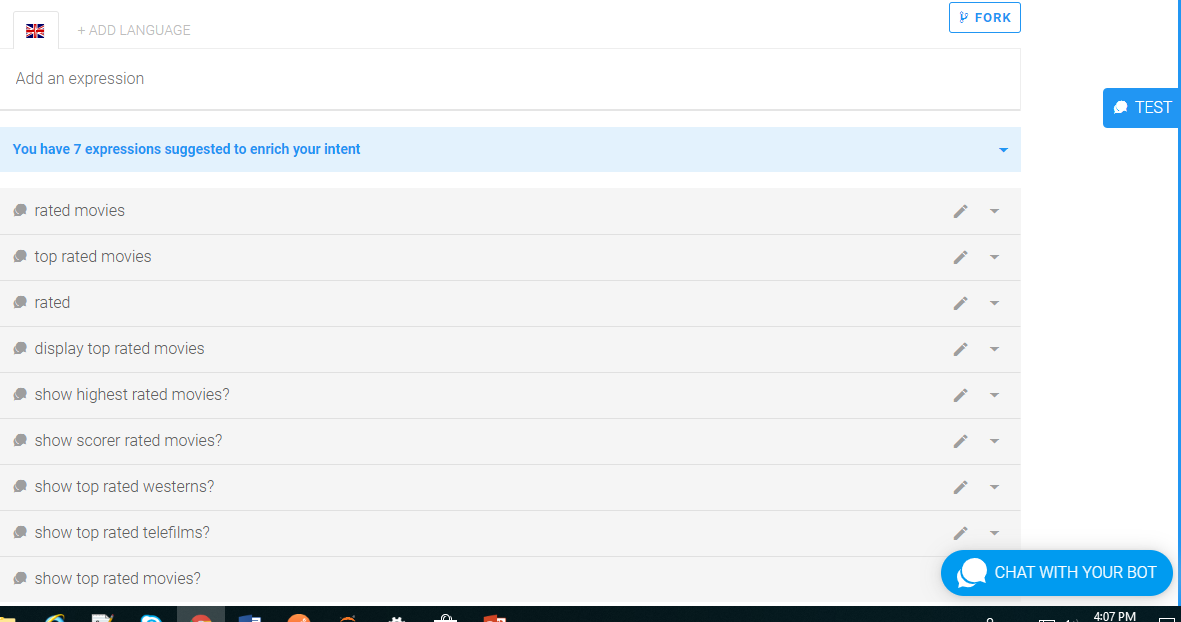
* 1. NLP using Recast.ai

In Recast.ai we build different skillset and its corresponding intents depending on the utterances and training data for the use case.

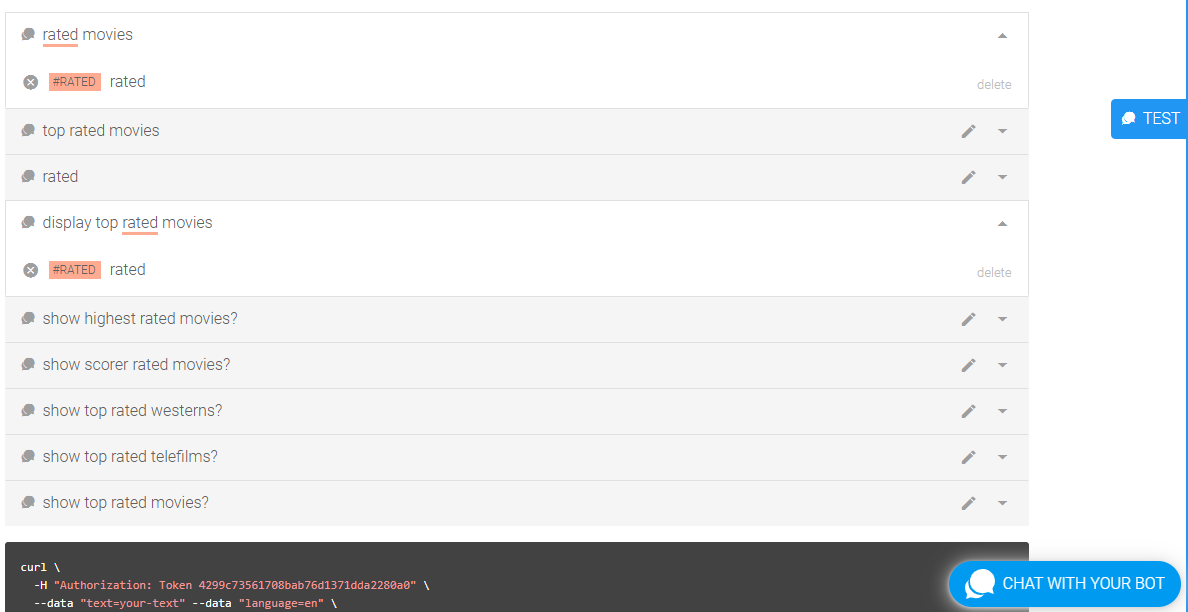




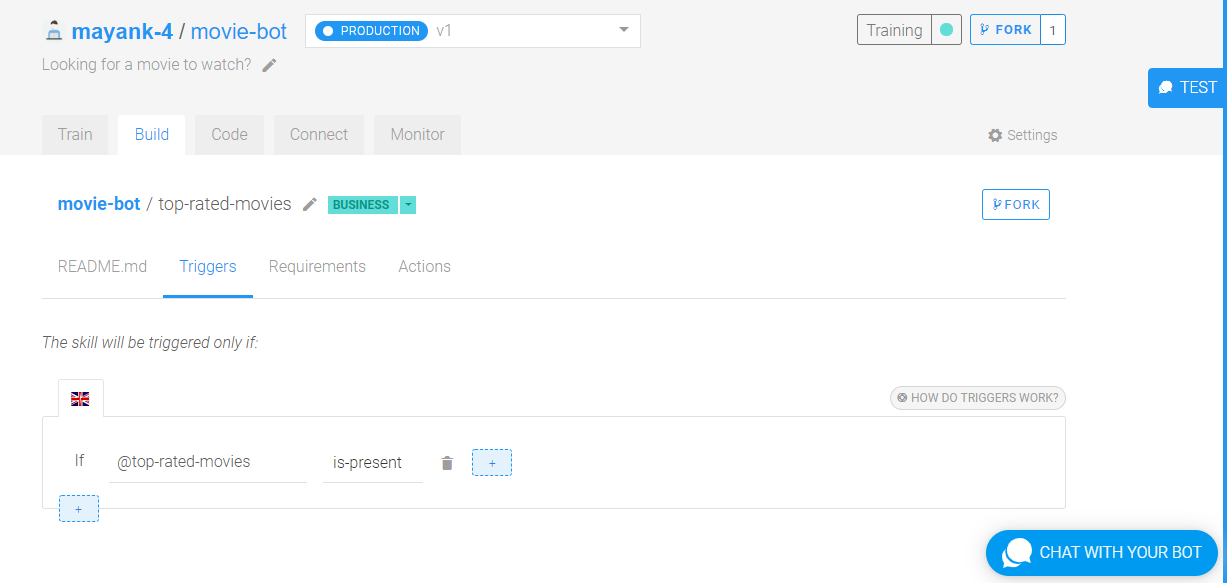
Based on the intents for each of the skill we train the model using different utterances. In the example below, we have different utterances to get top rated movies.



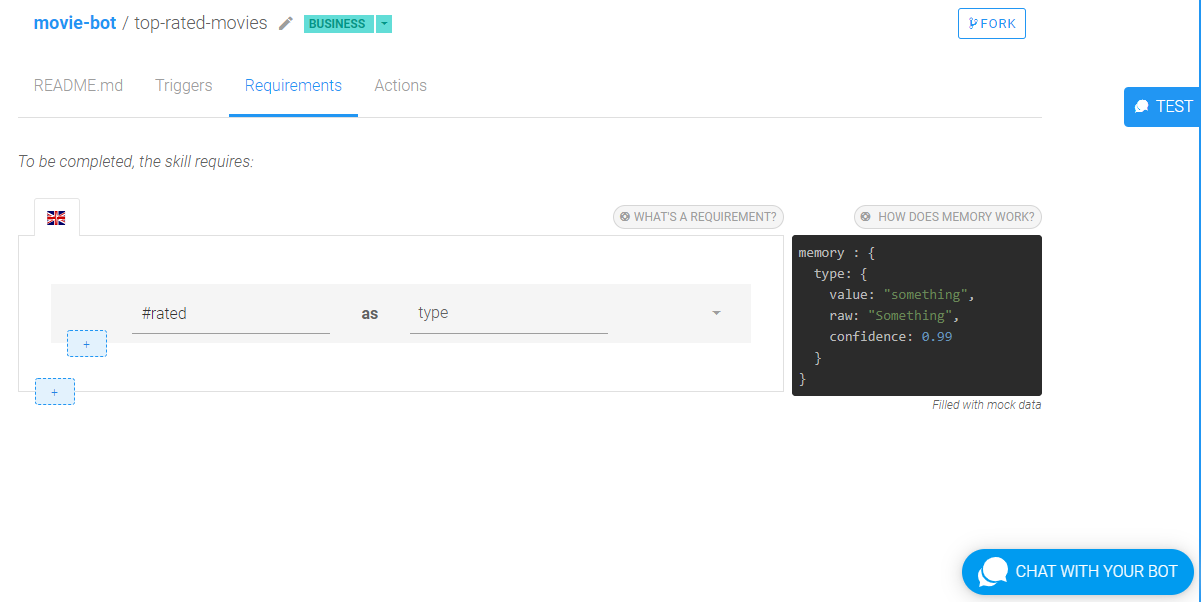
Based on the utterances, we map the keywords to entities. Example: rated maps to entity which is created to denote rated entity so that we can define trigger and action steps accordingly.



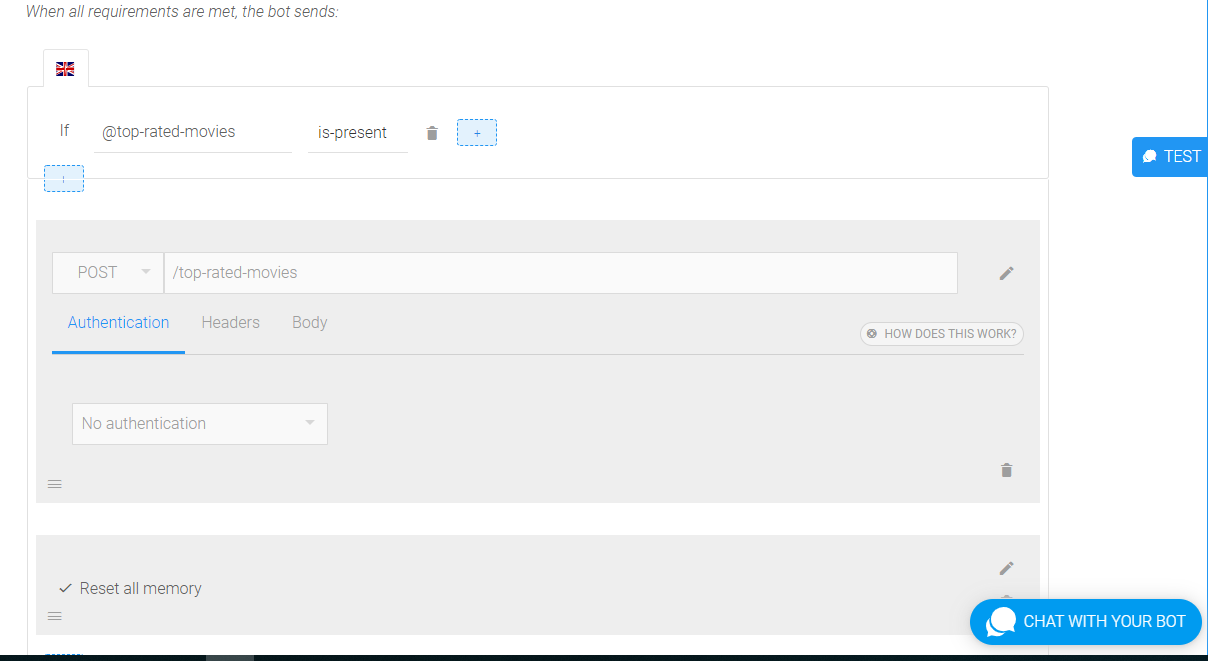
For each of the skills defined, we define the triggers, so that appropriate action can be defined



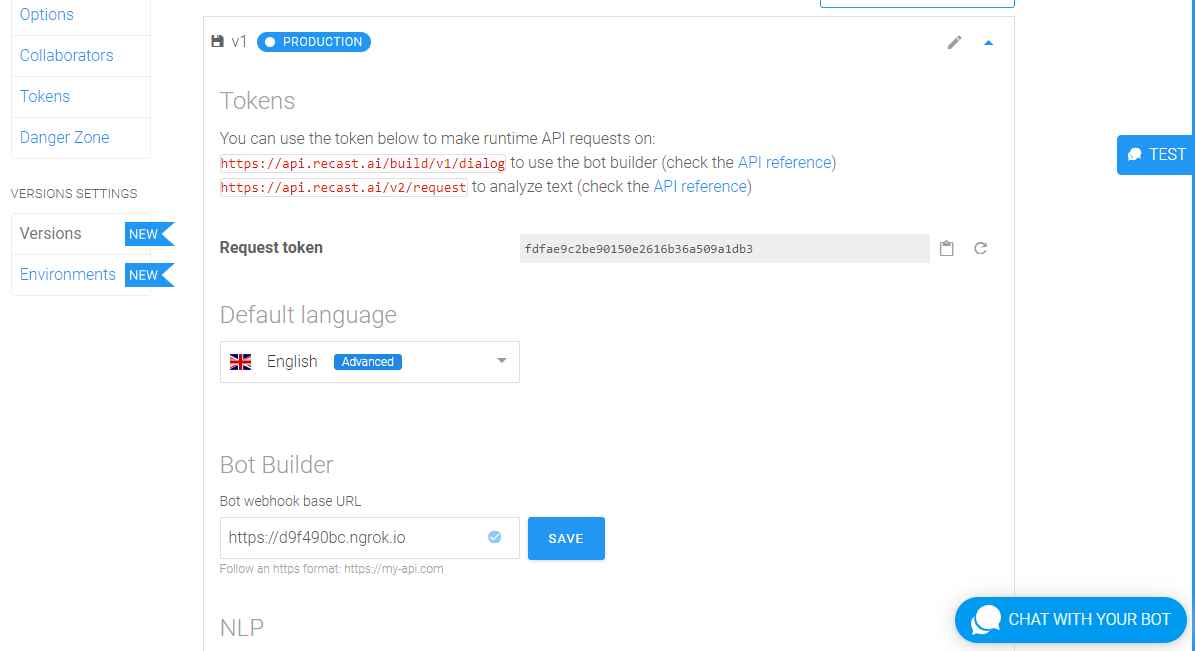
Requirements are optional to define inorder to call the appropriate action with correct data.



Action can be to show a sample response based on the intent detected like below and also it can be calling a webhook i.e. tmdb api through which tmdb api are called.



Adding ngrok url in the version settings of the bot in Recast.ai.



* 1. Recast.ai Skills

**get-movies**

This skill is developed for intents regarding discovering movies use case.

It will be triggered for intents get-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**popular-movies**

This skill is developed for intents regarding different categories use case.

It will be triggered for intents popular-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**top-rated-movies**

This skill is developed for intents regarding different categories use case.

It will be triggered for intents top-rated-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**upcoming-movies**

This skill is developed for intents regarding different categories use case.

It will be triggered for intents upcoming-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**nowplaying-movies**

This skill is developed for intents regarding different categories use case.

It will be triggered for intents nowplaying-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**review-movies**

This skill is developed for intents regarding movie reviews use case.

It will be triggered for intents review-movies.

Under Actions part of skill appropriate messages are added as response for each intent.

**release-date**

This skill is developed for intents regarding release date in different regions use case.

It will be triggered for intents release-date.

Under Actions part of skill appropriate messages are added as response for each intent.

**listgenre**

This skill is developed for intents regarding discovering movies use case.

It will be triggered for intents genre.

Under Actions part of skill appropriate messages are added as response for each intent.

**greetings**

This skill is developed for intents regarding Greetings use case.

It will be triggered for intents greetings.

Under Actions part of skill appropriate messages are added as response for each intent.

**goodbye**

This skill is developed for intents regarding Good-Bye use case.

It will be triggered for intents goodbye.

Under Actions part of skill appropriate messages are added as response for each intent

**fallback:**

This skill is triggered if user asks query not relating to any of the intents for which the bot is trained. It sends the message given under Actions part of skill as response.

1.4 NodeJS Layer

Node JS is used in ALEX as a middleware to fetch data from TMDB API and send the response to Recast. The API calls to TMDB API are made through Node JS and data retrieved is formatted according to the requirement and is sent to Recast.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

* + 1. Packages

The packages/modules used are express, body-parser, moment, request-promise.

**Express**: Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications.

**body-parser**: To handle HTTP POST request in Express.js version 4 and above, we need to install middleware module called body-parser. body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body.

1.4.2 Routes

**/discover-movies:**

In this route we fetch the kind such as movie or tv and the genre Id. These values are retrieved from the query given by user using **movie** entity. Then we make one API call such as GET call to tmdb api, here we pass kind dynamically in the URL. The response is stored in a variable and result set of movies is retrieved from the response and the reply is sent to recast.

API Used: axios.get(`[https://api.themoviedb.org/3/discover/${kind}`,](https://api.themoviedb.org/3/discover/$%7bkind%7d%60,%20) {

params: {

api\_key: config.MOVIEDB\_TOKEN,

sort\_by: 'popularity.desc',

include\_adult: false,

with\_genres: genreId, }, })

**/review-movies:**

In this route we fetch the movie name. These values are retrieved from the query given by user using **moviename** entity. Then we make two API call such as GET call to tmdb api, here we pass movie name dynamically in the URL and authorization token as header and also provide movie Id as a card in the second api call. The response is stored in a variable and result set of reviews is retrieved from the response and the reply is sent to recast.

API Used:

url1="[https://api.themoviedb.org/3/search/movie?api\_key=b079e3281956f4828176cf7ab807889e&query="+movieName](https://api.themoviedb.org/3/search/movie?api_key=b079e3281956f4828176cf7ab807889e&query=%22+movieName);

url2=”[https://api.themoviedb.org/3/movie/"+card+"/reviews?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1](https://api.themoviedb.org/3/movie/%22+card+%22/reviews?api_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1)”;

**/release-date:**

In this route we fetch the movie name. These values are retrieved from the query given by user using **moviename** entity. Then we make two API call such as GET call to tmdb api, here we pass movie name dynamically in the URL and authorization token as header and also provide movie Id as a card in the second api call. The response is stored in a variable and result set of release date is retrieved from the response and the reply is sent to recast.

API Used:

url1="[https://api.themoviedb.org/3/search/movie?api\_key=b079e3281956f4828176cf7ab807889e&query="+movieName](https://api.themoviedb.org/3/search/movie?api_key=b079e3281956f4828176cf7ab807889e&query=%22+movieName);

url2="[https://api.themoviedb.org/3/movie/"+card+"/release\_dates?api\_key=b079e3281956f4828176cf7ab807889e](https://api.themoviedb.org/3/movie/%22+card+%22/release_dates?api_key=b079e3281956f4828176cf7ab807889e)";

**/popular-movies:**

In this route we fetch the popular keyword. These values are retrieved from the query given by user using **popular** entity. Then we make one API call such as GET call to tmdb api, here we pass authorization token as header. The response is stored in a variable and result set of movies is retrieved from the response and the reply is sent to recast.

API Used:

url1="https://api.themoviedb.org/3/movie/popular?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1”;

**/top-rated-movies:**

In this route we fetch the rated keyword. These values are retrieved from the query given by user using **rated** entity. Then we make one API call such as GET call to tmdb api, here we pass authorization token as header. The response is stored in a variable and result set of movies is retrieved from the response and the reply is sent to recast.

API Used:

url1="https://api.themoviedb.org/3/movie/top\_rated?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1”;

**/upcoming-movies:**

In this route we fetch the popular keyword. These values are retrieved from the query given by user using **upcoming** entity. Then we make one API call such as GET call to tmdb api, here we pass authorization token as header. The response is stored in a variable and result set of movies is retrieved from the response and the reply is sent to recast.

API Used:

url1="https://api.themoviedb.org/3/movie/upcoming?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1”;

**/nowplaying-movies:**

In this route we fetch the popular keyword. These values are retrieved from the query given by user using **currentmovie** entity. Then we make one API call such as GET call to tmdb api, here we pass authorization token as header. The response is stored in a variable and result set of movies is retrieved from the response and the reply is sent to recast.

API Used:

url1="https://api.themoviedb.org/3/movie/now\_playing?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US&page=1”;

**/listgenre:**

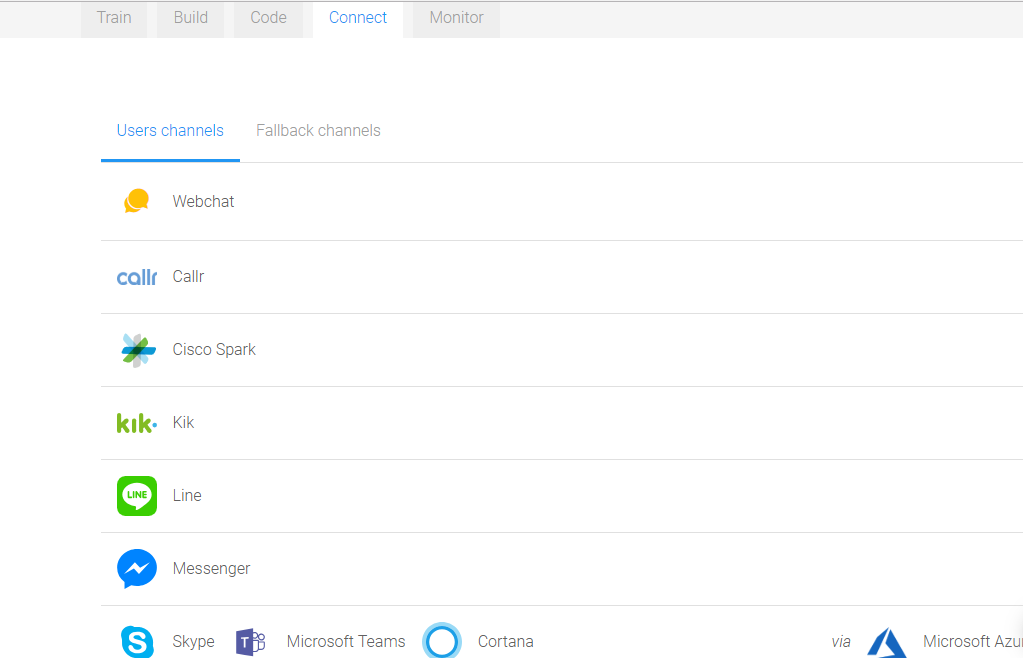
In this route we fetch the genre keyword. These values are retrieved from the query given by user using **genre** entity. Then we make one API call such as GET call to tmdb api, here we pass authorization token as header. The response is stored in a variable and result set of genre list is retrieved from the response and the reply is sent to recast.

API Used:

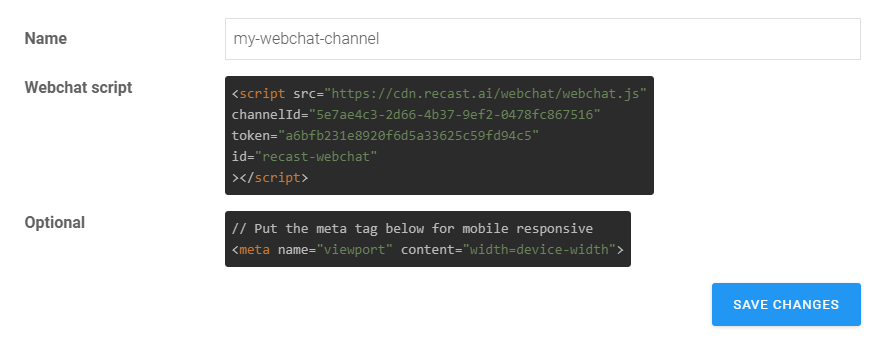
url1="https://api.themoviedb.org/3/genre/movie/list?api\_key=b079e3281956f4828176cf7ab807889e&language=en-US”;

* 1. WebChat Integration

The user channel through which user can connect with the bot can be configured. In Connect tab, we have different user channels like WebChat, Skype, Microsoft Teams, Facebook Messenger, Twitter, Line, Slack, Telegram etc.

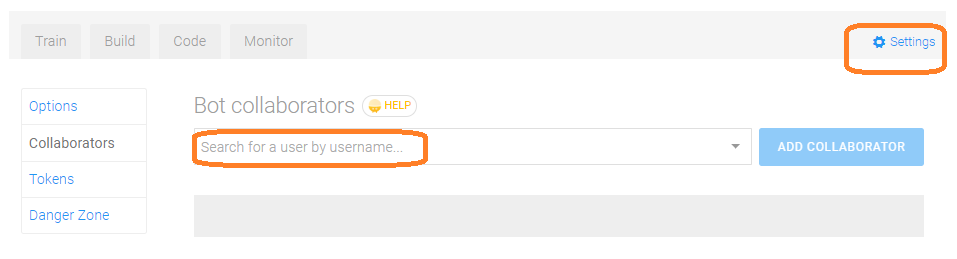


Create a new WebChat and copy paste the Webchat script url in the webpage you need to add the chatbot.



* 1. Deploy Recast.ai Bot

Add client’s Recast account as a collaborator and transfer the Bot.



* 1. References
* <https://recast.ai/>
* <https://www.themoviedb.org/>