**UMKC Student Helper Chatbot**

The basic idea of the project is to design an efficient chatbot that helps the students. The project involves several features like providing directions to different blocks in the campus, class schedule of the student, available timings of the advisors/faculty, various events happening in the campus.

**Project Stack:**

Below are the frameworks used to develop the Chatbot.

* Rasa Stack
* Angular
* Bootstrap (NgBootstrap)

**RASA Stack:**

**RASA stack is an open-source AI tool**and being an opensource framework, it is easy to customize. In fact, in many cases, Clients do not want to share their data and most of the tools available are cloud-based and provide software as a service. You can’t run them internally in your environment. So, we need to send your data to the third party. With RASA, there is no such issue. **We can build, deploy or host Rasa internally in our server or environment with complete control on it.**

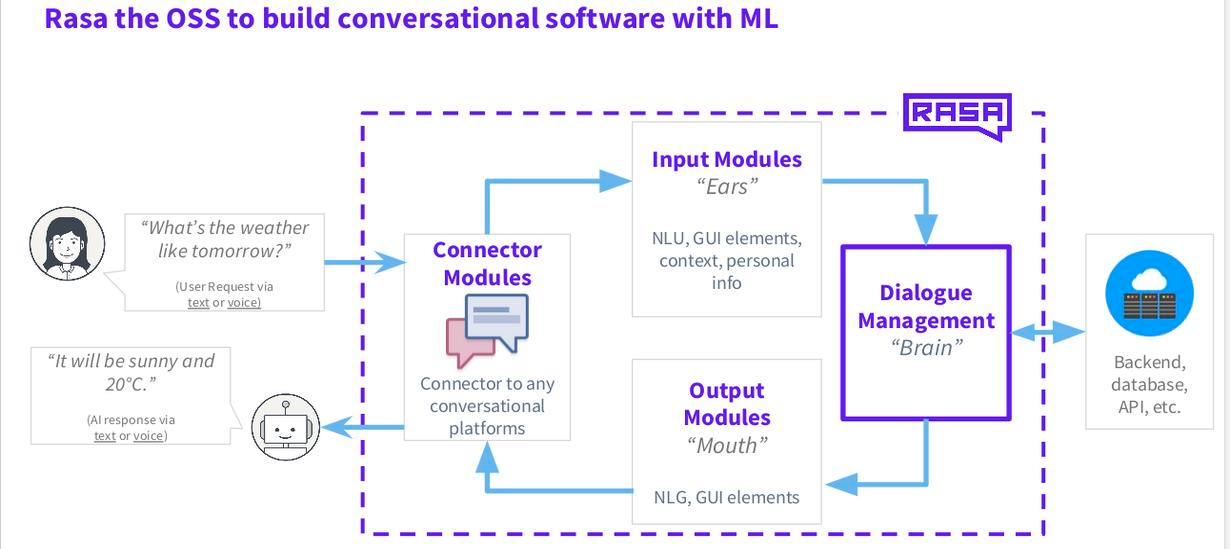
Rasa comes up with 2 components —

***Rasa NLU —***a *library* for natural language understanding (NLU) which does the classification of intent and extract the entity from the user input and helps bot to understand what the user is saying.

***Rasa Core —***a chatbot *framework* with machine learning-based dialogue management which takes the structured input from the NLU and predicts the **next best action** using a probabilistic model like LSTM neural network.

NLU and Core are independent, and one can use NLU without Core, and vice versa. Though Rasa recommends using both.

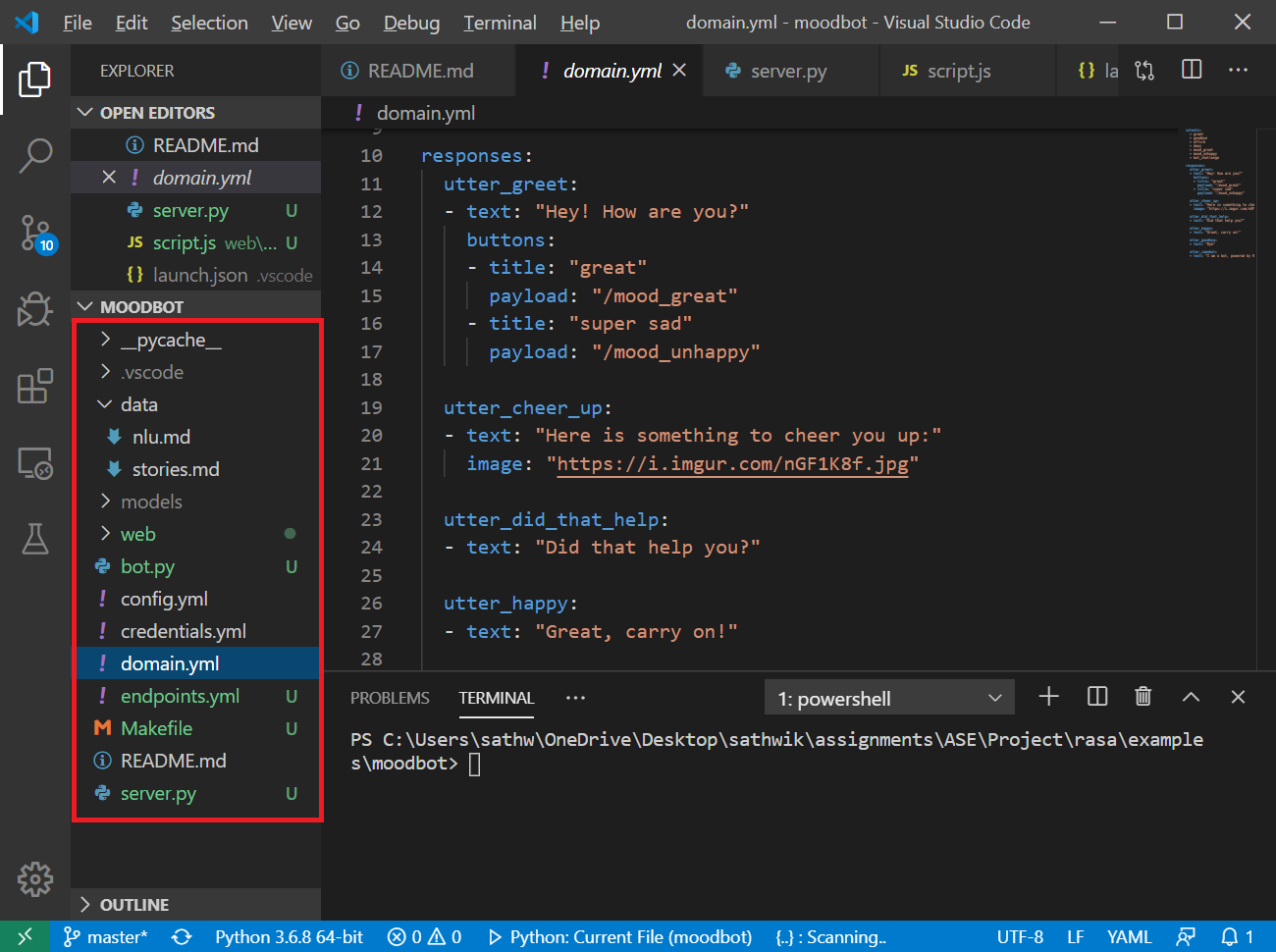
# Rasa architecture:



The following steps have been followed to create a chatbot agent in Rasa:

1. Install Rasa with pip
2. Initialize a New Rasa Project
3. Create NLU Training Data
4. Define Model Configuration
5. Create Stories
6. Define a Domain
7. Train the Model
8. Talk to the Assistant

# File Structure



|  |  |
| --- | --- |
| \_\_init\_\_.py | An empty file required to create a python package |
| actions.py | Has methods for custom actions |
| config.yml | Configuration file of NLU and Core models |
| credentials.yml | Connection details for integrating with other services |
| data/nlu.md | The NLU training data (Intents, Entities and Slots) |
| data/stories.md | The stories file (Intents, Actions and Flow) |
| domain.yml | List of all components |
| endpoints.yml | Endpoints details for connecting to channels |
| models/<timestamp>.tar.gz | The trained model file |

# data/nlu.md

**## intent:greet**

- hey

- hello

- hi

- hello there

- good morning

- good evening

**## intent:goodbye**

- good afternoon

- cu

- good by

- cee you later

- good night

- bye

**What is NLU.md?**

It contains the data required to train the bot – Intents, Entities and Slots.

* **Utterances**:

Things users say to the bot – sentences in natural language.

e.g: What are the hotels available near me?

* **Intents**:

Intents are the categories which the bot can identify / classify

e.g: What are the hotels available near me? 🡪 search\_hotels

* **Entities**:

Key information in an utterance used as arguments to extract answers.

e.g: What are the **hotels** available near **me**?

* **Slots**:

Slots are predefined buckets to control information capturing by the bot.

e.g: What are the **Indian**[*cuisine*] hotels available near me?

# data/stories.md

**## happy path               <!-- name of the story - just for debugging -->**

\* greet

  - utter\_greet

\* mood\_great               <!-- user utterance, in the following format: \* intent{"entity\_name": value} -->

  - utter\_happy

**## sad path 1               <!-- this is already the start of the next story -->**

\* greet

  - utter\_greet             <!-- action of the bot to execute -->

\* mood\_unhappy

  - utter\_cheer\_up

  - utter\_did\_that\_help

\* affirm

  - utter\_happy

What is Stories.md?

* Stories file is used to specify different (if not all possible) flows the user may go through during the conversation with the chatbot.
* It specifies the actions to be taken (or) the templates to send as utterances to the user at each turn.

# config.yml

language: en

pipeline:

  - name: "SpacyNLP"

  - name: "SpacyTokenizer"

  - name: "SpacyFeaturizer"

  - name: "RegexFeaturizer"

  - name: "LexicalSyntacticFeaturizer"

  - name: "DIETClassifier"

    epochs: 100

  - name: "EntitySynonymMapper"

policies:

  - name: TEDPolicy

    max\_history: 5

    epochs: 100

  - name: MemoizationPolicy

  - name: MappingPolicy

# Endpoints.yml

action\_endpoint:

  url: "http://localhost:5055/webhook"

# domain.yml

intents:

  - greet

  - goodbye

  - affirm

responses:

  utter\_greet:

  - text: "Hey! How are you?"

    buttons:

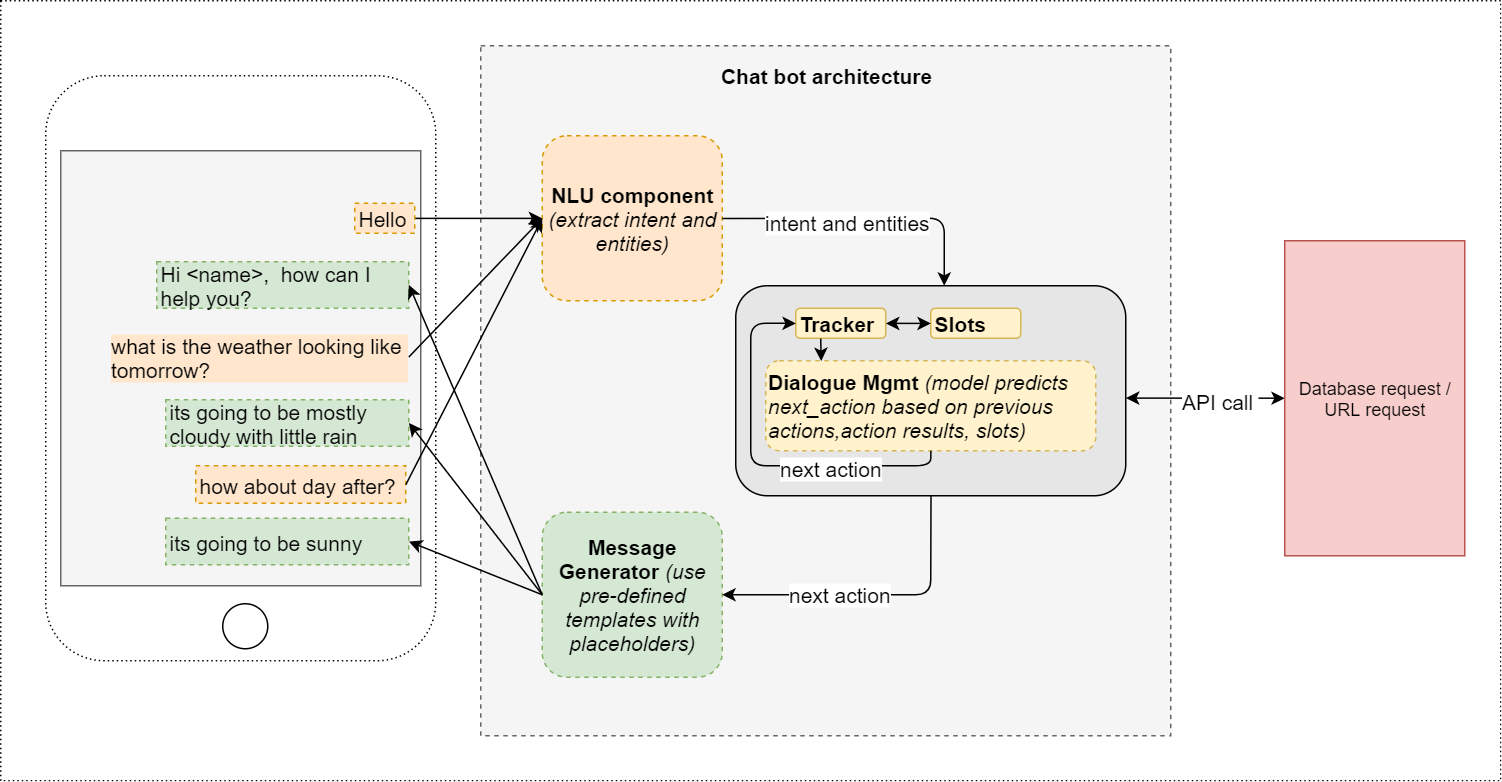
    - title: "great"

      payload: "/mood\_great"

    - title: "super sad"

      payload: "/mood\_unhappy"

Putting it together…



**Angular:**

**Services:**

We use Angular framework in the frontend to send the messages entered by user to the bot and also to fetch the bot’s response and display to user. To serve this purpose we make use of Angular HttpClient service.

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable()

export class BotDataService {

  public botApiUrl = '<rasa webhost url>'

  constructor(private http:HttpClient) { }

  public sendMessage(message: any) {

    // http POST request to send the user message

    this.http.post(this.botApiUrl, message);

  }

  public getResponse(response: any) {

    // http POST request to send the user message

    this.http.get(this.botApiUrl);

  }

}

**Router:**

Based on the user query, the bot can respond in several ways, for example a normal text response, an image or any hyperlink(url). Except for url format we display the messages directly in the chat window to the user.

If bot responds to the user query with any url, we make use of angular routing in order to navigate to that page in the background.

Angular Router enables us to build Single Page Applications with multiple views and allow navigation between these views.

The following are the essential parts of angular router.

**Router-Outlet:** It is a directive provided by routing library. Router inserts the component that gets matched based on the current page url which is provided by the bot.

<router-outlet></router-outlet>

**Routes and Paths:** We configure different routes in Application Routing Module (app.routing.module).

import { NgModule, Component } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { ScheduleComponent } from './pages/schedule/schedule.component';

const routes: Routes = [

    {

        path: 'schedule',

        Component: ScheduleComponent

    }

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

**Bootstrap:**

In order to make the chatbot application as a responsive once, we make use of Bootstrap.

Bootstrap is the most popular HTML, CSS, and JavaScript framework for web front-end development. It’s great for developing responsive, mobile-first web sites.

**Installing Bootstrap into Angular:**

Once we create a angular application for our chatbot, we can import the bootstrap as a npm package.

npm install bootstrap jquery –save

This installs Bootstrap and jQuery into the node\_modules folder within the project directory

Once the packages are installed we need to add the file paths in the angular-cli.json under styles array.

"styles": [

    "styles.css",

    "../node\_modules/bootstrap/dist/css/bootstrap.min.css"

  ],

  "scripts": [

    "../node\_modules/jquery/dist/jquery.min.js",

    "../node\_modules/bootstrap/dist/js/bootstrap.min.js"

  ],

**Ng-Bootstrap:**

In addition to bootstrap, we will make use of Ng-Bootstrap module which contains a set of native Angular directives based on Bootstrap’s markup and CSS. It is also available as a npm package and can be installed using following command.

npm install --save @ng-bootstrap/ng-bootstrap

Once installed you need to import Ng-Bootstrap’s main module NgbModule from the package @ng-bootstrap/ng-bootstrap.

Add the following import statement to app.module.ts:

import {NgbModule} from '@ng-bootstrap/ng-bootstrap';

Once we import the NgbModule, we can make use of the built-in components of ng bootstrap like Accordion, Alert, Buttons, Carousel, Collapse etc.

# Github location for project:

<https://github.com/srichakradhar/CS5551-Team1-StudentBot.git>

**Next Steps:**

1. Create a SPA to represent the functionality in the college website.
2. Attach the chatbot component to the page.
3. Integrate the bot with rasa server with webhook.
4. Connect the bot to google and other APIs.

Task tracking on ZenHub:

