GOOGLE SUMMER OF CODE 2021

SCORE LAB CHATBOT PROJECT PROPOSAL

Project Introduction (SCoRe Lab Gitter ChatBot)

The Scorelab chatbot project aims to welcome newcomers to the score lab's Gitter channel. Quickly respond to their queries to get them started with the community projects based on their skills and interests. The existing members can also ask for any required information from the chatbot about any project. The project also targets the detection of spam messages and removes the spammer from the room. After that, the chatbot will store the user's username in the database to avoid future threats.

The chatbot should warn a user on detecting any offensive message from them that violates the community guidelines. After a specific number of warnings, the chatbot would take strict actions to avoid future threats. The chatbot project is totally new and has the following expectations:

- ➔ Once a new user joins the channel and sends a welcome message, the Chabot Should welcome the user to the community.
- ➔ If this user provides details on what he/she is interested in, query a data source and provide feedback with relevant projects, startup guides, documentation links etc.
- ➔ If an existing user tries to ask a question related to a project that the score-bot can answer, answer those questions.
- ➔ If any user tries to spam the channel with offensive messages, record them in a database and automatically remove them.
- ➔ If any user tries to communicate with another contributor disrespectfully, warn them with a message and record them in a database.

The score-lab-bot idea essentially has three components: one is the chatbot account on Gitter/Github that will be the face of score lab chatbot, the second is the application server (Node.js) to listen and respond to the messages on the Gitter chatroom and lastly, a backend API server (Flask) to process the client's message, store and serve all the required information needed by the chatbot to answer the queries. As this project needs to be built from scratch, all testing is done on a test chatbot. The current version of the test chatbot can perform some core functionalities of the application server, which include:

- ★ Listening to every message event on the **stream.gitter.im** API.
- ★ Extract the message by parsing the JSON string on the message event.
- ★ Processing the message and checking for the newcomer use case.
- ★ Welcome the newcomer to the community on their first message in the chatroom.
- ★ Extract all necessary information from the message of the client, like username and skillset.

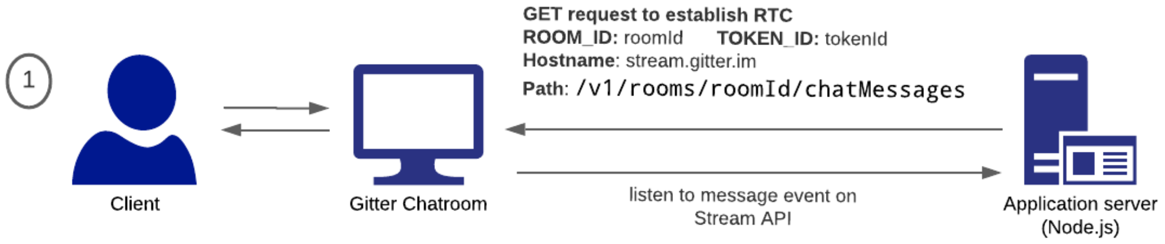
These achievements almost complete the work on the application server. Continuing with this, my tasks during the summers would essentially involve writing the complete backend from scratch, possibly in Flask. The backend mainly includes Analysis of the messages, REST API and performing CRUD on DB.

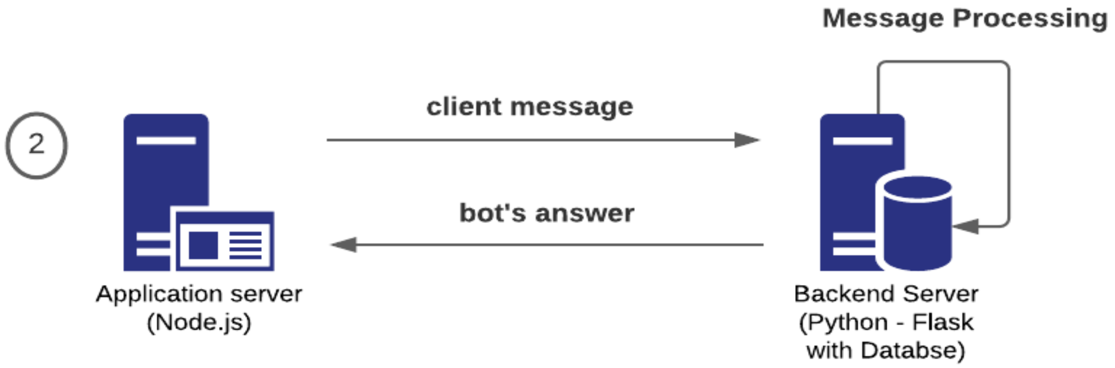
Using Flask and Python adds the following benefits to the project:

- ❏ As Flask is lightweight, it is an excellent choice for end-pointing, REST-ful API and querying a DB.
- ❏ Some python data types and string methods make processing of the text more accessible.
- ❏ A variety of ML packages and libraries can be used to improve identifying a use case.

The design of the project is entirely based on use cases and expected outcomes. My milestones in the creation of the chatbot include:

- Establish a connection with the chatroom:
    - Create a function call to invoke an HTTPS connection request to the Gitter Streaming API (The streaming API allows real-time access to rooms).
    - Use the response object to extract the text message for every message event in the streaming API.


1
Client
Gitter Chatroom
GET request to establish RTC
ROOM_ID: roomId TOKEN_ID: tokenId
Hostname: stream.gitter.im
Path: /v1/rooms/roomId/chatMessages
listen to message event on Stream API
Application server (Node.js)


2
Application server (Node.js)
client message
bot's answer
Message Processing
Backend Server (Python - Flask with Databse)

- Recognise new contributor on their first message, and create a chatbot's response:
    - Their messages follow a similar pattern, i.e. introduction + skills + interests.
    - Make a python model to store some keywords used by newcomers. Process every message in the chatroom and check for the keywords.
    - Post a welcome message with a general startup guide and chatbot help instructions.
    - Create a method to get the skills or interest of the client. Process them to make a standard list of tech stack.
    - Fetch the required data from the database based on the tech stack, and create a message with that information.
    - Reply (@mention) to the client with the final message and post it on the Gitter server.
- Handle the queries of an existing user:
    - Existing users use a specific format like "@mention_bot <query_type> <project_name>" to ask questions to the chatbot about any project.

    - QUERY_TYPE is an identifier of what exactly to show for a given PROJECT_NAME.
    - Query the database and create a message object, including the query result and send it to the chatroom.
- Make a database and store few models init:
    - Decide the data types, tables and structure the columns. Collect the data from the existing score projects and store their details inside the DB.
- Write the complete backend from scratch:
    - Use **SQLAlchemy** (the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL) to query the python server's database.
    - Use Flask for end-pointing purposes.
    - Use **Flask-RESTful,** which is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with existing ORM/libraries. Flask-RESTful encourages best practices with minimal setup.
- Write test cases covering the complete codebase:
    - Test the Node server as well as the Flask server with manual testing.
- Create proper documentation for the project:
    - Complete the user guide on using, installation and deployment.
    - The proper API documentation for allowing easy integration with the Github/Gitter account of the chatbot.
    - Wiki documentation for setting up the project using docker.
    - Use tools like DOxygen and Sphinx to complete the documentation.
- Deploying a demo on the Heroku cloud platform:
    - Deploy the application and backend servers using Github on Heroku.
    - Containerizing the app using Docker and uploading it to the container directory.
    - Deploy the container image to Cloud Run.
- More Milestones:
    - Link other sources of information:
        - Research on Github API to connect it with the chatbot if required. The chatbot will be able to post a message for new PRs and issues.

- Track offensive messages:
    - The chatbot will use a separate model to recognise the offensive messages.
    - Create a default warning message object, and Post it to the channel.
- Establish Spam protection:
    - Detect spam messages and record the username of the spammer in the database.
    - Remove the user from the room.
- Use machine learning packages to identify the use cases:
    - Use python tools in the backend to perform supervised learning
    - Write some sample data sets containing the different messages a newcomer sends.
    - Use high accuracy ML libraries offered by python.
    - Deploy the new project patch to either GCP or Heroku cloud.

Project Goals

- ❏ Structure the complete app, decide the tech and overview It's control-flow.
- ❏ Plan a public profile for the chatbot on platforms like GitHub and Gitter.
- ❏ Decide the database, python tools for database migrations and the directory structure of the flask backend.
- ❏ Identify and collect the type of information required to store in the database.
- ❏ Create an application server in node.js.
- ❏ Build a backend API server using Flask to make a REST-ful API.
- ❏ Do deep research to implement ML, aiming to improve the recognition of a newcomer.
- ❏ Brainstorm on bot's answer format for different types of queries.
- ❏ Create or use an existing chatroom and integrate the chatbot with it.
- ❏ Deploy on Heroku cloud.
- ❏ **Writing resourceful documentation for the chatbot project to help the users make their chat rooms intelligent and fast**.
- ❏ Completing the documentation about the installation and deployment on **Heroku cloud**.

- I will also publish a blog monthly/weekly sharing my experience with score lab and GSoC.

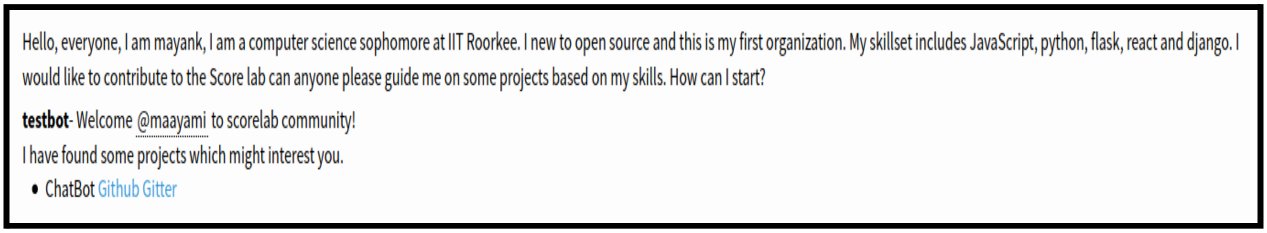
Proof of Concept

I have successfully implemented the core use cases of the project idea with my approach, and I am fully confident that this proposed solution strongly forms the basis of all the project idea requirements.

Here is a list of all the successfully implemented tasks:

- Listen to all the messages in the target chatroom (with ROOM_ID).
- Recognize a new contributor from their initial chat messages.
- Welcome the newcomer with a default startup guide to the community projects.
- Fetch all the required details provided by the newcomer to guide them to the open-source contribution.
- Solve a specific query asked by any existing member.

Here are some screenshots of the chatroom in which the bot is active:

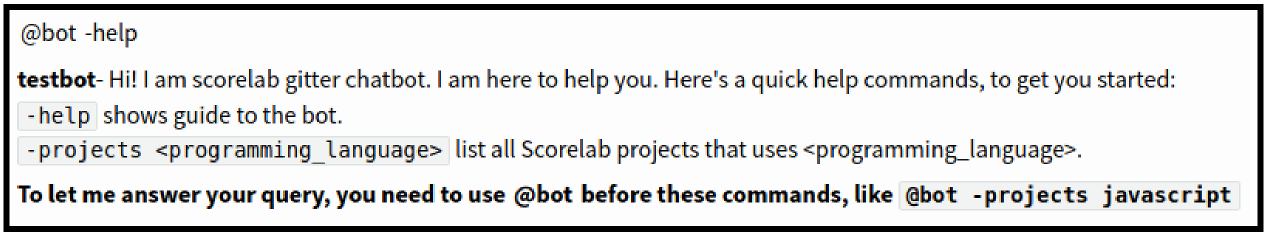


Hello, everyone, I am mayank, I am a computer science sophomore at IIT Roorkee. I new to open source and this is my first organization. My skillset includes JavaScript, python, flask, react and django. I would like to contribute to the Score lab can anyone please guide me on some projects based on my skills. How can I start?

testbot- Welcome @maayami to scorelab community!

I have found some projects which might interest you.

- ChatBot Github Gitter



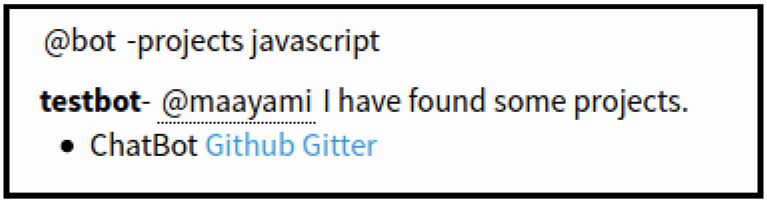
@bot -help

testbot- Hi! I am scorelab gitter chatbot. I am here to help you. Here's a quick help commands, to get you started:

`-help` shows guide to the bot.

`-projects <programming_language>` list all Scorelab projects that uses <programming_language>.

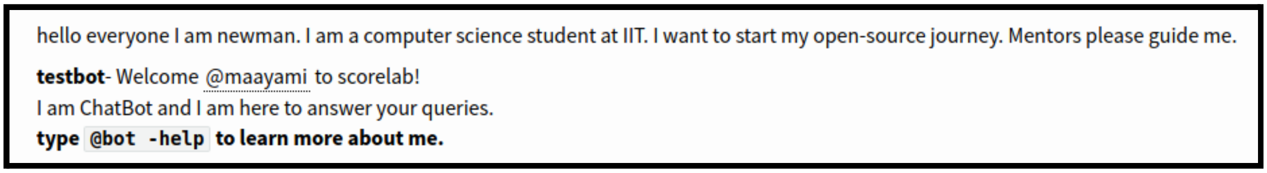
To let me answer your query, you need to use @bot before these commands, like `@bot -projects javascript`



@bot -projects javascript

testbot- @maayami I have found some projects.

- ChatBot Github Gitter



hello everyone I am newman. I am a computer science student at IIT. I want to start my open-source journey. Mentors please guide me.

testbot- Welcome @maayami to scorelab!
I am ChatBot and I am here to answer your queries.
type `@bot -help` to learn more about me.

Implementation

1. Decide the structure of the complete application:

The application design mainly has two parts covering the app's entire control flow: The application server and the backend API.

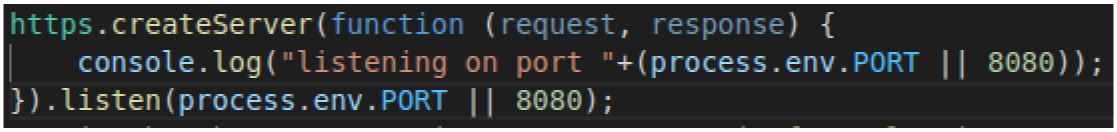
2. The face of the chatbot:

The Gitter API requires a token, which could be generated after registering the application to a Gitter developer account. The application also needs to have a Gitter account. No doubt that will be the public profile of the chatbot.

3. Create the Application server:

Starting with `npm init`, the project's initial file structure includes package.json, package-lock.json and a .env file. The .env file contains the required config variables such as **ROOM_ID** and **TOKEN**, while Heroku requires these config variables explicitly during the deployment process. Keeping that in mind, I have used npm package **dotenv** to read those config variables.

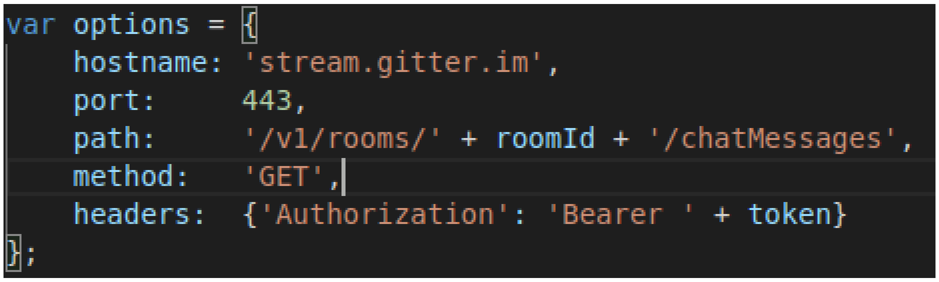
a. I will use **HTTPS** to initialize the server on any available **PORT**. Like in the below code, the HTTPS method **createServer** was called to bind the PORT on the **Heroku cloud server**.



```
https.createServer(function (request, response) {
    console.log("listening on port "+(process.env.PORT || 8080));
}).listen(process.env.PORT || 8080);
```

b. Implement a function that fetches Gitter account details of the chatbot, bypassing the TOKEN. It will be used to get the **username** of the chatbot and the **ROOM_ID** of the chatroom.

c. To establish a connection between the chatroom and the chatbot, I have created a GET request to the streaming API (**stream.gitter.im**) with the following request object.



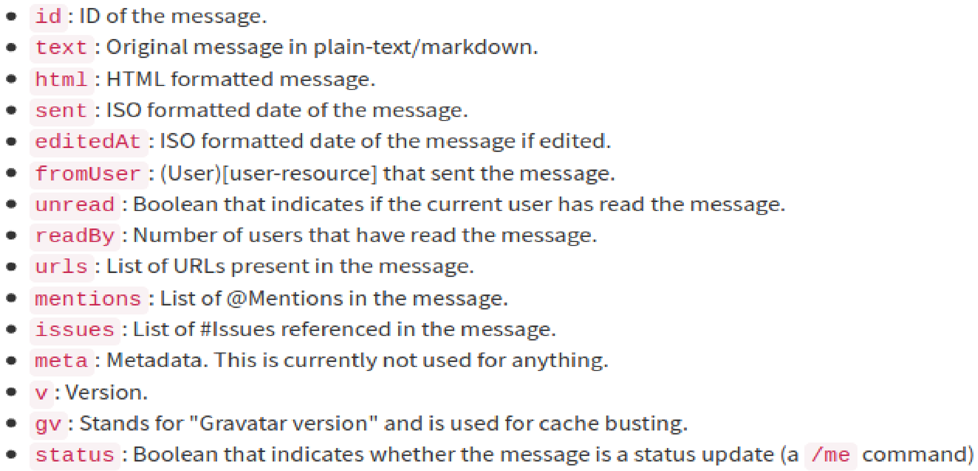
```
var options = {
    hostname: 'stream.gitter.im',
    port:     443,
    path:     '/v1/rooms/' + roomId + '/chatMessages',
    method:   'GET',
    headers:  {'Authorization': 'Bearer ' + token}
};
```

Chatbot **Uses the streaming API to listen to messages, events and changes in rooms.** The streaming API allows real-time access to rooms. Unlike other APIs, all requests to the streaming API should be made to *stream.gitter.im*.

To connect to the Streaming API, form an HTTP request and consume the resulting stream for as long as is practical. The Gitter servers will hold the connection open indefinitely (barring a server-side error, excessive client-side lag, network hiccups or routine server maintenance).

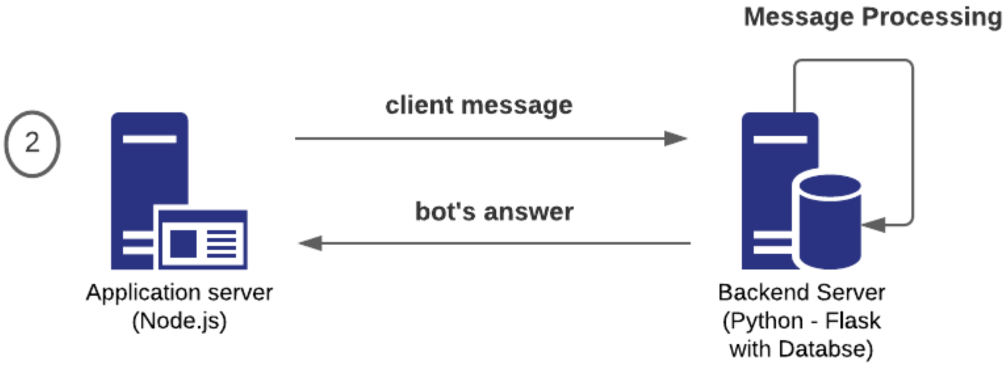
d. Parse the **JSON response** to convert it into a message object. The message object contains the following parameters:

The chatbot will need **text** and **fromUser** from the above object.



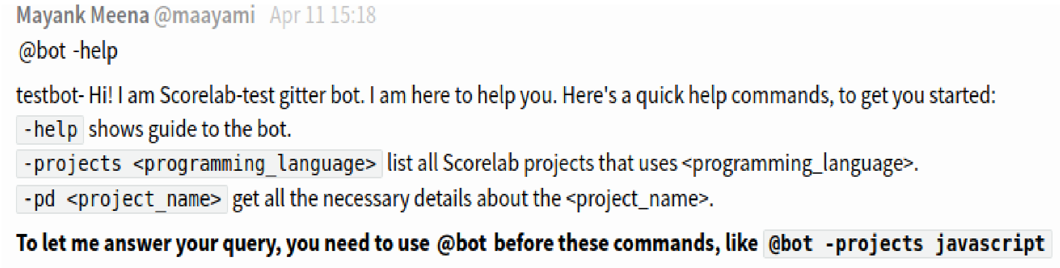
- `id` : ID of the message.
- `text` : Original message in plain-text/markdown.
- `html` : HTML formatted message.
- `sent` : ISO formatted date of the message.
- `editedAt` : ISO formatted date of the message if edited.
- `fromUser` : (User)[user-resource] that sent the message.
- `unread` : Boolean that indicates if the current user has read the message.
- `readBy` : Number of users that have read the message.
- `urls` : List of URLs present in the message.
- `mentions` : List of @Mentions in the message.
- `issues` : List of #Issues referenced in the message.
- `meta` : Metadata. This is currently not used for anything.
- `v` : Version.
- `gv` : Stands for "Gravatar version" and is used for cache busting.
- `status` : Boolean that indicates whether the message is a status update (a `/me` command)

e. Connect to the python server.



f. Send the message text to the Python server that processes the message and returns a response containing the chatbot's answer. Create a GET request with the message inside a query string of the request qs: {q: clientMessage}.

g. Write some independent answers such as chatbot help instructions, startup guide and a default answer. The default answer will be used if the python server doesn't give any response. The startup guide will help the newcomers to know more about the community and Its projects. Write an answer that will contain chatbot help instructions. An example use case is shown below:



Mayank Meena @maayami Apr 11 15:18

@bot -help

testbot- Hi! I am Scorelab-test gitter bot. I am here to help you. Here's a quick help commands, to get you started:

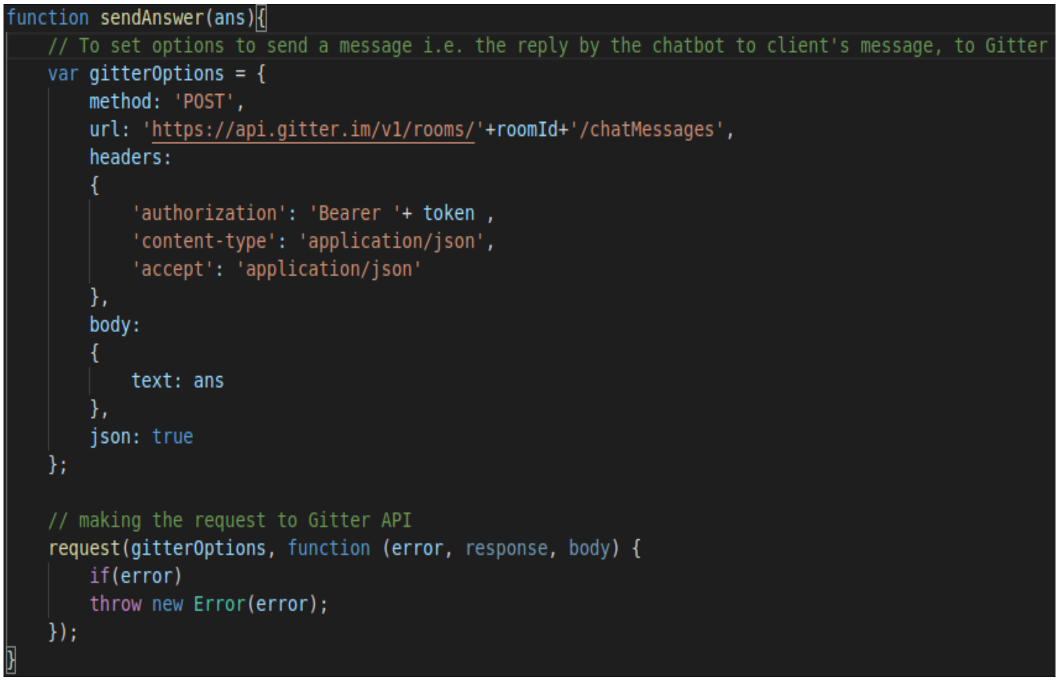
`-help` shows guide to the bot.

`-projects <programming_language>` list all Scorelab projects that uses <programming_language>.

`-pd <project_name>` get all the necessary details about the <project_name>.

To let me answer your query, you need to use @bot before these commands, like `@bot -projects javascript`

h. After receiving the chatbot's answer, I modified the text by adding @**mention** before it. Finally, make a POST request to the Gitter API to send the message.



```
function sendAnswer(ans){
    // To set options to send a message i.e. the reply by the chatbot to client's message, to Gitter
    var gitterOptions = {
        method: 'POST',
        url: 'https://api.gitter.im/v1/rooms/'+roomId+'/chatMessages',
        headers:
        {
            'authorization': 'Bearer '+ token ,
            'content-type': 'application/json',
            'accept': 'application/json'
        },
        body:
        {
            text: ans
        },
        json: true
    };

    // making the request to Gitter API
    request(gitterOptions, function (error, response, body) {
        if(error)
        throw new Error(error);
    });
}
```

NOTE: The .env file will be ignored by Github. A .env.sample will be in the source code to let the users know about its format.

4. Build the backend API (Python server):

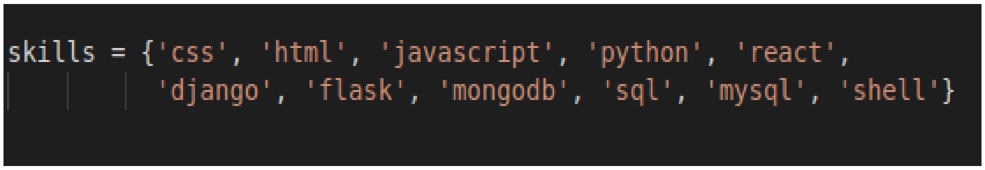
a. I have decided to go with the **Postgres** database. To make migrations in a SQL database with the flask backend, I will use the python library **SQLALchemy**. It is a Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

b. The database will store information about the community projects, including Programming languages, a link to the Gitter channel, and a link to the Github repository.

c. Make a REST-ful API in the flask backend using **Flask-RESTful** to handle the API end-pointing. Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with the existing ORM/libraries.

I have used flask restful_packages such as **Resource**, **Api** to successfully implement the REST API in the backend. The **GET** request made by the application server contains the message of the user inside a query string. To get the message from it, I have used flask_restful package **reqparse,** which adds the query params to an args object.

d. **Message processing** is an essential part of this project. It could be implemented in many ways, but the initial development would include:
   i. The initial queries asked by the new contributors follow a similar pattern -> introduction + skills + interests. They usually mention their abilities and ask the community to guide them to some projects.
   ii. I will create a **set** containing the **keywords** mainly used by the new contributors and loop over each word in the message, checking or those keywords., The chatbot also looks for the skills inside the message. I have used the python sets to store the keywords.



```
skills = {'css', 'html', 'javascript', 'python', 'react',
          'django', 'flask', 'mongodb', 'sql', 'mysql', 'shell'}
```

e. **Existing Users** ask their queries using a template command message -projects <programming_language>. I will first extract the parameters <query_type> and <query>. Then using-switch case, I would code their actions.

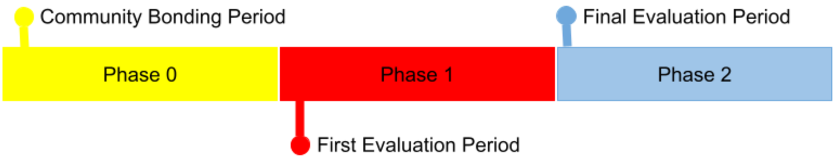
5. Integrate the chatbot with the chatroom:

a. The requirement is to have a Gitter and a Github account. It allows the chatbot to make requests on the Gitter REST API and the Stream API.
b. This will need two things - a **ROOM_ID** and a **TOKEN** key. Then update these config variables inside the .env file with their respective keys.
c. We need these config variables during the deployment process to add them explicitly to the Heroku Cloud Platform.

6. Containerization, Dockerization, Documentation and Deployment of the project

The application server and the backend API needs to be deployed on Heroku to function the chatbot all the time. I will dockerize the app to handle the dependency issues in future.
I also aim to write complete documentation on the project covering user guide, installation guide, and steps to deploy the project on the Heroku Cloud Platform.

Timeline



Timeframe	Start Date - End Date	Task
Week 00	17th May - 7th June	• Discuss the project possibilities and requirements of the initial setup with the mentors. • Finalize the tech stack and project milestones with the mentors. • Study similar projects on Github. • Discuss different use cases with the mentors. • Decide the use cases to implement. • Discuss the testing methods with mentors. • Explore the tech stack.
Week 01	8th June - 14th June	• Setup project repositories on Github. • Setup Gitter channel for testing and to track the project progress. • Prepare the initial development setup. • Decide the endpoints, API, database, cloud platforms.
Week 02	15th June - 21st June	• Start building the application server. • Research about the other helpful APIs which

		can be used in the chatbot.
Week 03	22nd June - 28th June	• Start building the backend API. • Research and collect information about the community projects. • Create the Flask server. • Connect the database. • Establish REST-ful API. • Create all endpoints
Week 04	29th June - 5th July	• Study all the required python tools. • Make methods to implement all the use cases in the python server. • Connect the application server with the backend API.
Week 05	6th July - 12th July	At the end of phase I, the following features will be completed: ★ Identifying a newcomer. ★ Welcome the newcomer and guide him with the community projects. ★ Answer all the queries of existing members. ★ Warn a user on sending an offensive message.
Week 06	13th July - 19th July	• Deploy the project components to the Heroku cloud platform. • Connect the Gitter channel. • Run manual testing covering entire use cases. • Test both servers for the large data set.
Week 07	20th July - 26th July	• Containerize the project components. • Deploy the containers on the Heroku cloud. • Write complete documentation for the project.
Week 08	27th July - 2nd August	• Improvise the message sent from the chatbot. • Discuss the activation of the chatbot to Scorelab's main Gitter channel.
Week 09	3rd August - 9th August	• Learn the required tools to implement other use cases. • Complete the other use cases.
Week 10	10th August - 16th August	• Fix bugs and polish the new features and review the documentation. • Prepare for the final evaluation.

After August 16

If I will be able to deliver all of the above mentioned milestones in time, I will move ahead to implementing the extra milestones that I have mentioned. After this, I will proceed with the future goals of the project.

Personal Information

Basic Information

Name: Mayank Meena
Gender: Male (he/him/his)
Nationality: India | IST (UTC + 5:30)

Contact Information

Email: mayank_m@cs.iitr.ac.in
Phone: +917427834626
Website: mayankmeena.web

Educational Information

Institute: IIT Roorkee
Degree: B.Tech
Major: Computer Science, 2nd-year
Graduation: 2023

Social media

Github: maayami
Linkedin: Mayank Meena

Q. Few sentences about the student and why he/she thinks as the best person to do this project?

Since my freshman year, I have been exploring coding and software development. I have learned a variety of programming languages and created many open-source softwares to contribute to society. I have a good understanding of the requirements of this project idea. I have good experience working in a similar development environment. I have been contributing to this project for a while now and I fully understand its need. I have already implemented the core use cases of the idea on a test chatbot and would like to contribute more during the GSoC period. I am consistently working on this project idea and have a complete understanding of the main theme and future requirements.

SCoRe Contributions

I am an active user of Scorelab's Gitter channel and contributing to the community by helping newcomers to get started with the community's projects based on their interest and skills.

References

- Test Bot Application Server Code - Github
- Test Bot Backend Server Code - Github
- Test Bot Chatroom - Gitter
- Gitter API - Documentation
- Github API - Documentation