

Project Report -Group 30



TEAM MEMBERS

<u>NAME</u>	<u>SJSU ID</u>
Abhishek Madan	011408969
Nachiket Joshi	011408956
Sagar Mane	011419135
Saurabh Gedam	011451674

TABLE OF CONTENTS

<u>CHAPTER</u>	<u>PAGE</u>
CHAPTER 1 – Slackbot Creation Framework.....	3
CHAPTER 2 – Project Idea	6
CHAPTER 3 – Architecture.....	8
CHAPTER 4 – USER INTERFACE FLOW	9
CHAPTER 5 – APPLICATION SCREENSHOTS	11
CHAPTER 6 – USECASES.....	15
CHAPTER 7 – CHALLENGES AND FEATURES.....	16
CHAPTER 8 – FUTURE SCOPE FOR THE PROJECT.....	20
CHAPTER 9 – CONCLUSION.....	21
CHAPTER 10 – REFERENCES	22

SLACKBOT CREATION FRAMEWORK

• INTRODUCTION:

ChatBots are on a rise. These are automated computer programs, which respond to user queries by employing a suitable ML algorithm to gauge what the user is asking and reply appropriately. Bots have appeal and huge fan following not only among the developer community, but also among the non-developer tech community. People can now exploit the easy integration of their bots with the chat platforms like, Facebook, Slack or Tinder to automate certain tasks, thereby saving time or increasing productivity.

Let us get to know terminologies first,

• Slack :

Slack is a cloud bases team collaboration tool. Almost every programming team nowadays uses two tools one is git and the other is slack. It won't be an understatement to say that there are two types of people in this world: those who have never heard of slack and those who cannot even imagine their life without it.

The main reason behind this can be perceived is that the slack is a chat app specifically designed for business purpose rather than individuals. The slack also distinguishes itself from the competition in the extent to which it lets users customize it to meet their needs. This customization means the use of programmable bots that serve many mundane or complex tasks and the ease with which that can be done.

The tasks that can be done through the slack are link/embed social media posts, download images and even store them, and 'slackbot' can be set up to answer common questions or perform complex or mundane tasks automatically.

At its peak, Slack wants to be the repository for everything (just like git) a company uses, tying together disparate apps like Dropbox, Google Docs, Twitter and many others. Users come for the chat and cumulatively contribute for the project, and stay because everything else works so much better through the app and the bots that again the teammates themselves can design.

The positive point being the bots can be designed for the specific use of the team or they can even be commercialized. I mean you can even form a slack team to commercialize a slack bot and that is the beauty of it.

- **Bot User:**

Bot users and their human counterparts have almost similar properties and prominently can be listed as follows;

1. They exist in the team directory,
2. They have profile photos, names, and bios,
3. They can be direct messaged or mentioned,
4. They can post messages and upload files,
5. They can be invited to and kicked out of channels and private groups.

One of the striking and biggest difference between regular users and bot users is that instead of interacting with a team via one of Slack's mobile or desktop apps, bot users are controlled programmatically via a bot user token that accesses one or more of Slack's APIs. This token is individual and very important and needs to be kept secured.

Another difference, which actually makes them BOTS, is that the Bot users can't "log in," in a traditional way they don't have a password, and they only have access to a subset of all of the API methods available to regular users.

The amazing thing about the bots is that within a Slack channel, bot users can do nearly anything you can program them to do.

Slack has two different kinds of bot users: custom bots and app bots. Each of them serves a different purpose and offers different functionality.

This bot can be designed in many languages and there are many packages available which will make your life easier when it comes to designing a bot.

• HEROKU

Heroku is a cloud Platform-as-a-Service (PaaS) app.

This supports several programming languages that are used as a web application deployment model.

The benefits of using Heroku to deploy our bots can be listed as follows,

1. PAAS
2. Documentation is very good. All the APIs are elaborately explained and readily available.
3. Have in built tools and architecture and does not need much extra enhancements.
4. Limited control over architecture while designing app.
5. Heroku is best at what they provide as a service.
6. Deployment is taken care (through git commands only) and no need to do extra programming.
7. Good and ready support.
8. Not time consuming.
9. Very simple debugging and elaborate logging is available.
10. Simple tool belt gives a git like feeling.
11. Currently Heroku runs on AWS instance thus our EC2 container will be fast to respond to users which we have demonstrated through Jmeter testing

PROJECT IDEA:

Our idea is simple, we have to give a platform for rookie developers to develop their Slack Bots and abstract almost all the additional complex steps that one needs to take in order to host that bot on a cloud and make it live.

In order to integrate a bot with a channel on Slack, there are a number of configuration steps involved such as,

- Creating a web app project module
- Creating a cloud based repository to hold the source code
- Run the application on cloud so that the bot is active independent of the developer's machine state.

Our idea is to provide a framework, which would assist a novice person with coding a bot, guide him through the process of setting up of the environment and deploy the bot on the cloud all abstracted to a single button click.

By using our framework, which would be a hosted service (EC2), the user can focus more on the behavioral part of the bot and less on the configuration part.

With the help of our framework the goal was to abstract following tasks,

1. Requirement of a traditional development tool (viz. Eclipse)
2. Requirement of knowledge of all the requirements needed to make the bot live on cloud
3. Requirement of a debugging environment
4. Requirement of keeping into account the dependencies of a node package.
5. Requirement of compiler and syntax checks.
6. Requirement of maintaining cloud directories by giving all the basic operational APIs available in the framework.
7. Requirement of deleting app, rebuilding it and pushing it in the same place. (in short versioning facility for a user.)
8. Establishing connection with your slack and cloud without making any involvement of a user.

- **COMPETITIVE LANDSCAPE:**

There are only a few web applications, which provide user with such a framework. Moreover these platforms are too strict with the bot behavior coding guidelines. They do not allow developers to include additional, rich packages of the underlying language. Looking at how a bot could be helpful for a person, a more liberal framework can definitely find its place in this competitive market.

- **TECHNOLOGIES USED:**

- **Front End:**

- 1) HTML
- 2) CSS
- 3) JavaScript
- 4) jQuery
- 5) AngularJS

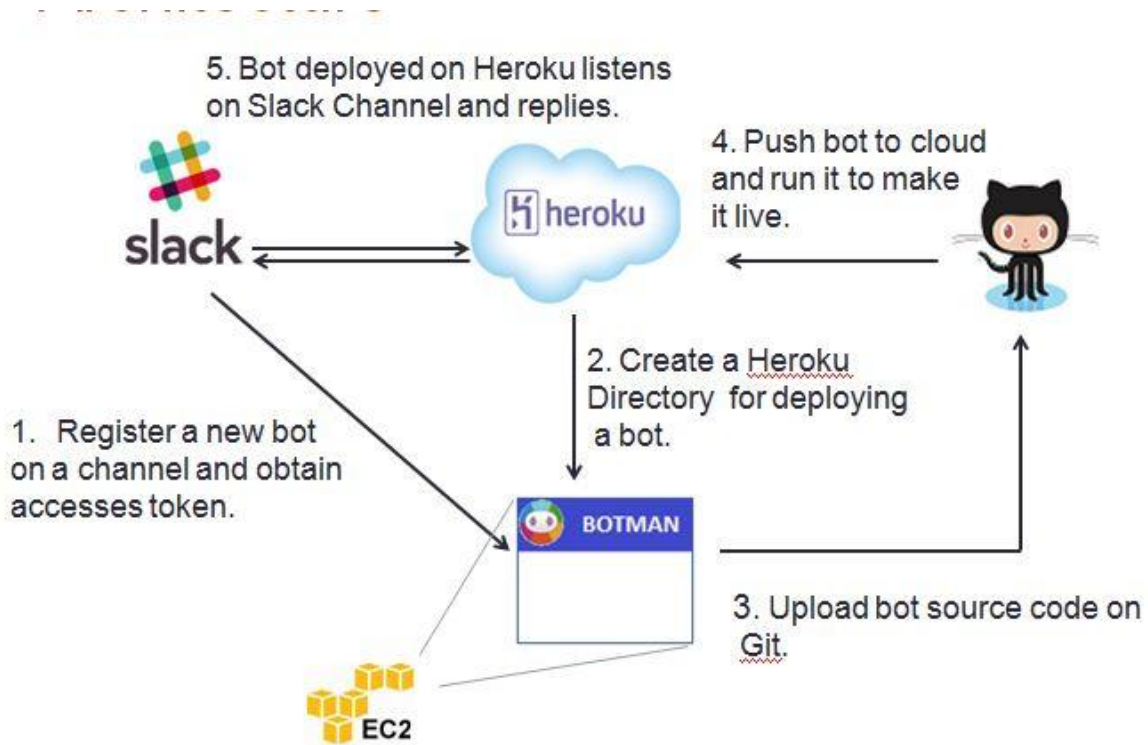
- **Server Side:**

- 1) Node package manager
- 2) nodeJS

- **Platform For Storage And Deployment:**

- 1) **GitHub** : It holds the code for the hosting the node application bot.
- 2) **Heroku** : It is a cloud service for hosting the bot.

ARCHITECTURE:

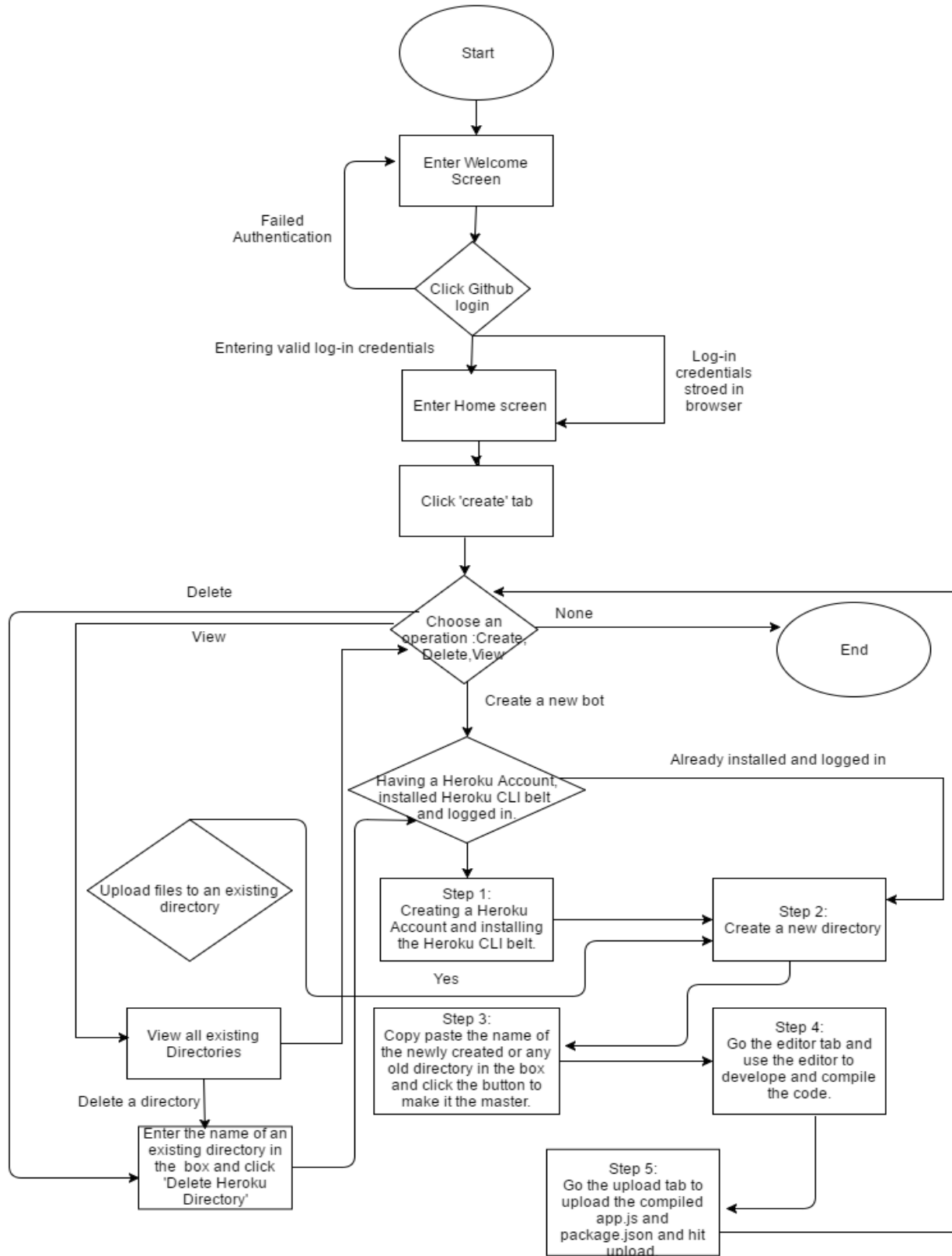


USER INTERFACE FLOW:

Our user interface for the framework can be easily understood with the help of this following diagram.

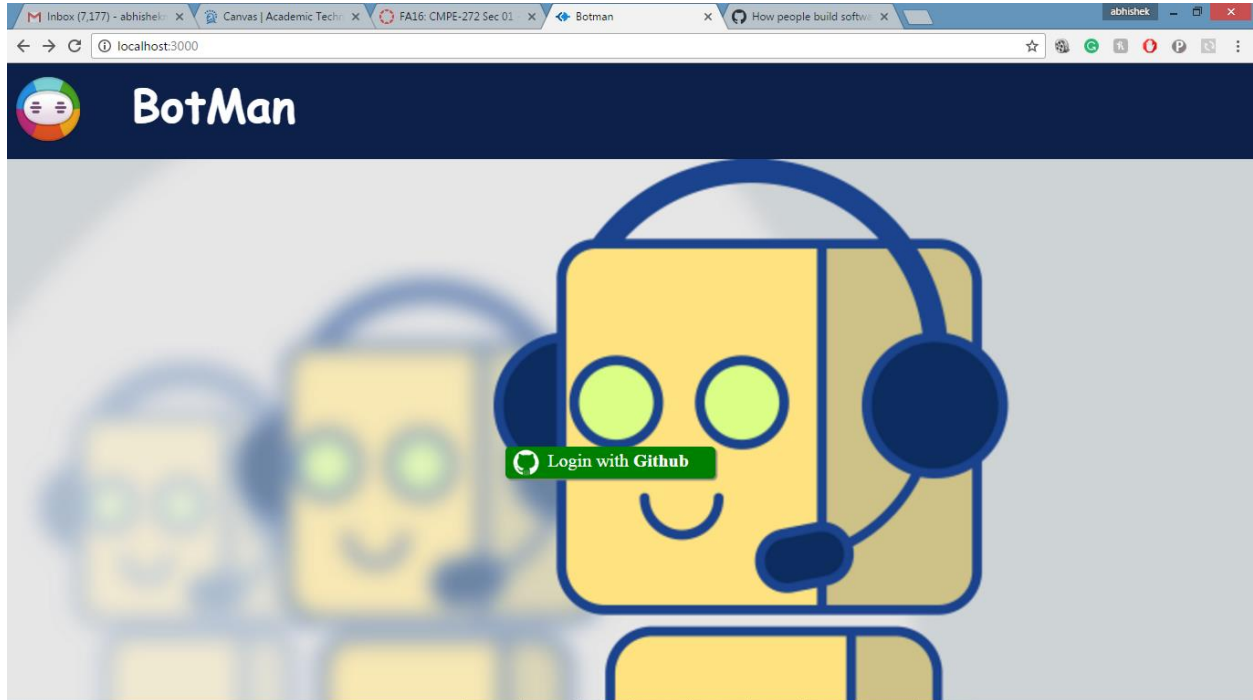
- 1) A user logs in to the system.
- 2) Uses basic commands to create a directory to create and push bots
- 3) Maintains coherence in directories by using VIEW and DELETE commands.
- 4) Use the JavaScript environment that we have created to check the syntaxes in your code.
- 5) Use '**save workspace**' facility to switch between tabs and maintain integrity of your data.
- 6) Use 'Upload' button to finally make your bot alive and check results.

CMPE 272 – Project Report – Team 30

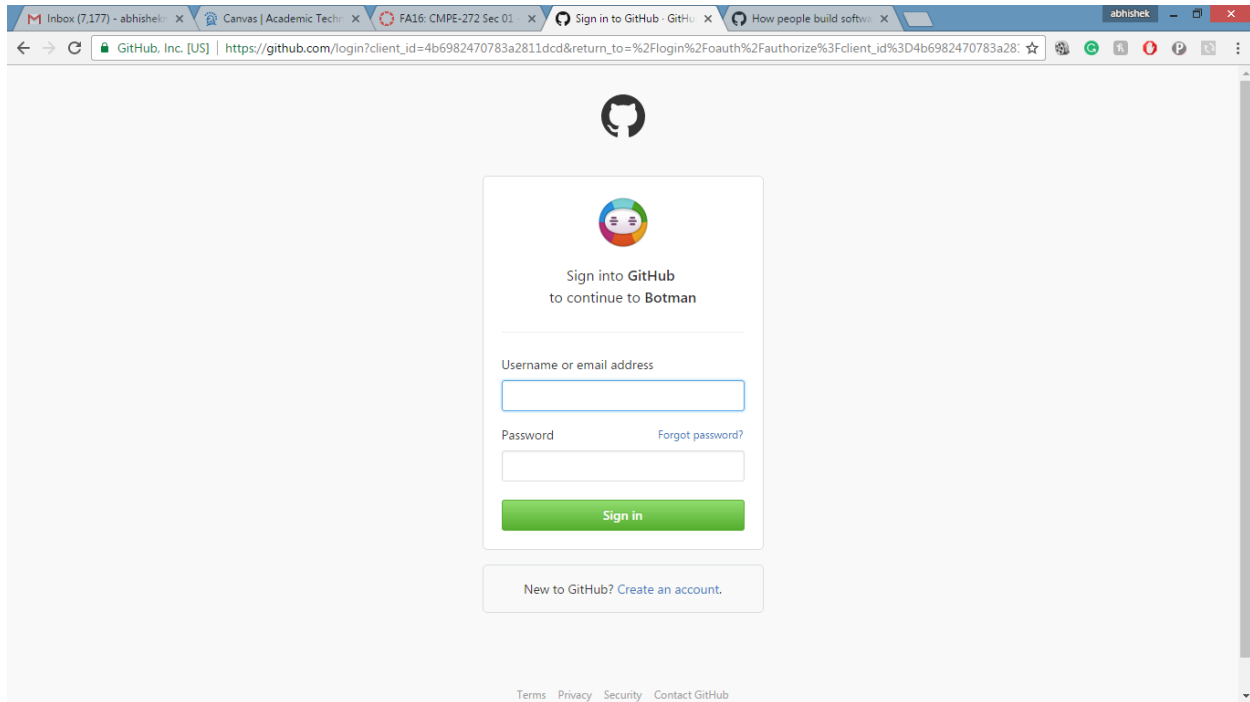


APPLICATION SCREENSHOTS:

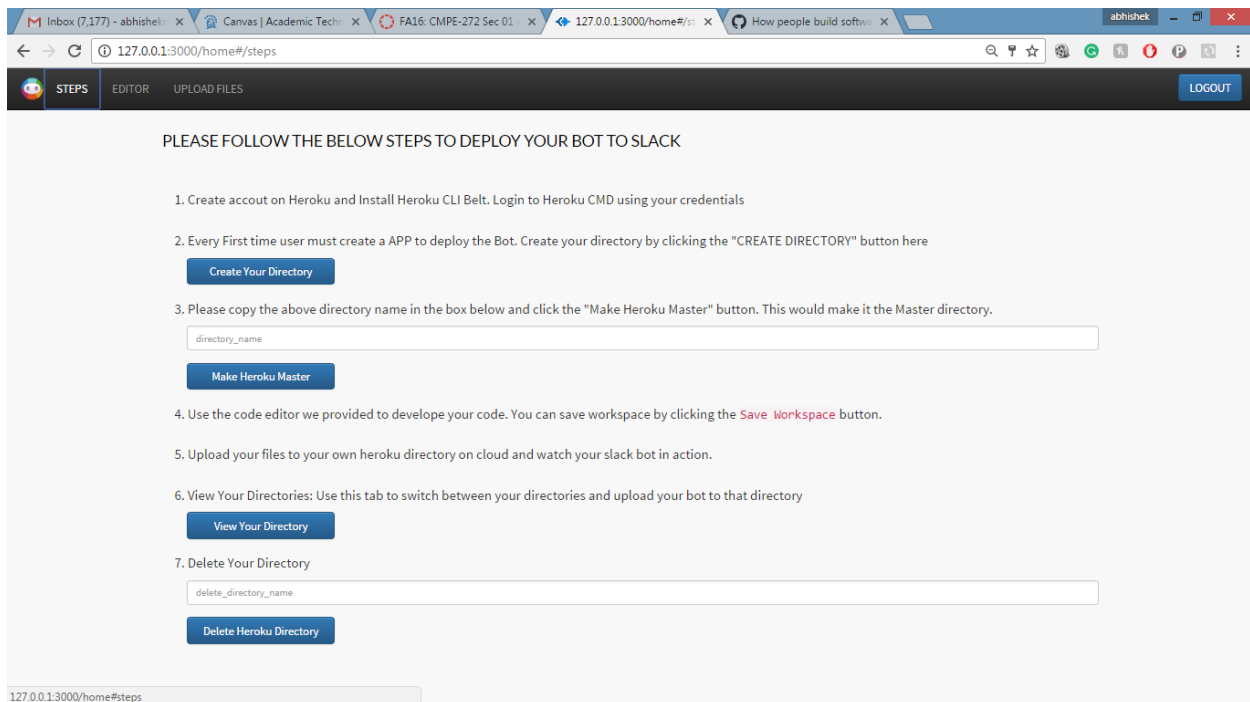
1) Landing Homepage



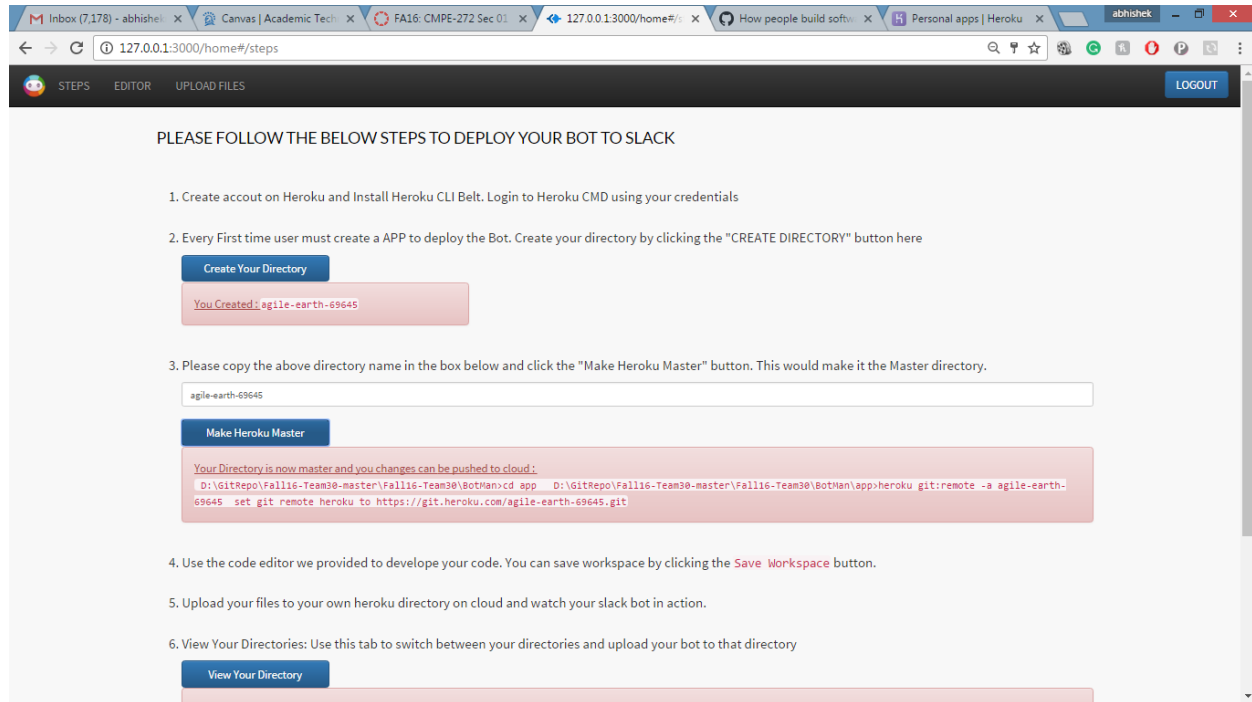
2) GITHUB LOGIN REQUIRED



3) Simple API to maintain the coherence of your bots



4) APIS in ACTION



PLEASE FOLLOW THE BELOW STEPS TO DEPLOY YOUR BOT TO SLACK

1. Create account on Heroku and Install Heroku CLI Belt. Login to Heroku CMD using your credentials
2. Every First time user must create a APP to deploy the Bot. Create your directory by clicking the "CREATE DIRECTORY" button here

Create Your Directory

You Created agile-earth-69645
3. Please copy the above directory name in the box below and click the "Make Heroku Master" button. This would make it the Master directory.

agile-earth-69645

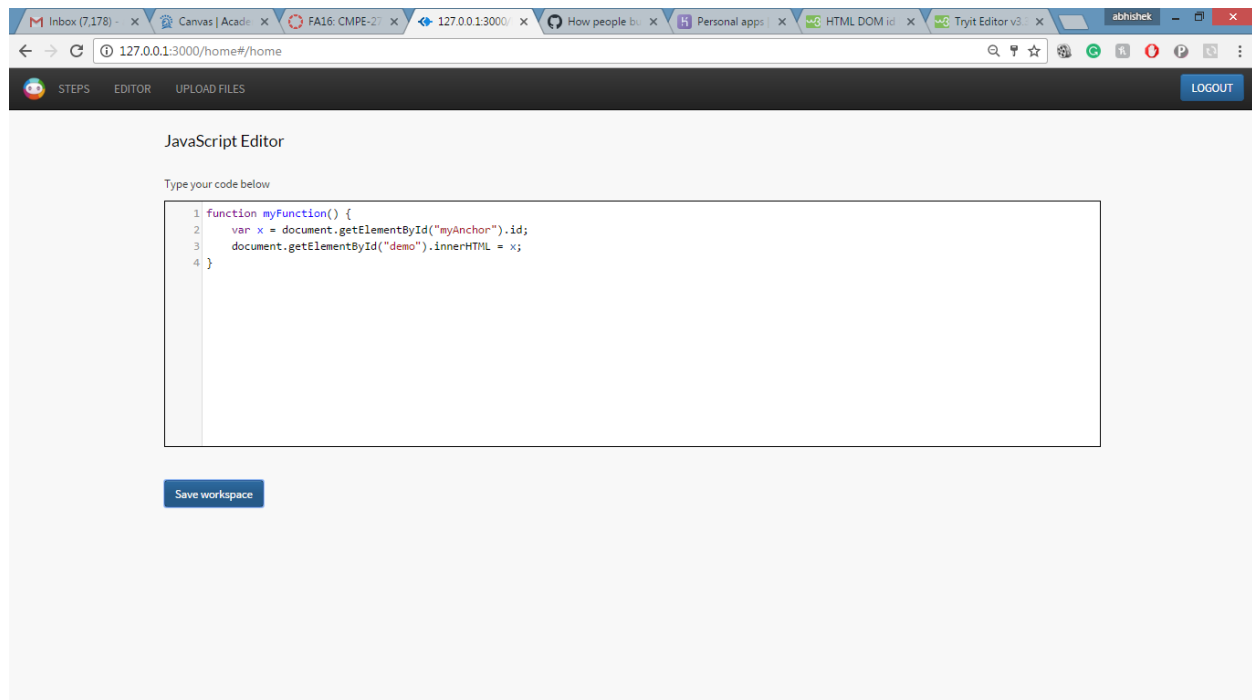
Make Heroku Master

Your Directory is now master and you changes can be pushed to cloud:

```
D:\GitRepo\Fall16-Team30-master\Fall16-Team30\BotMan>cd app D:\GitRepo\Fall16-Team30-master\Fall16-Team30\BotMan\app>heroku git:remote -a agile-earth-69645 set git remote heroku to https://git.heroku.com/agile-earth-69645.git
```
4. Use the code editor we provided to develop your code. You can save workspace by clicking the [Save Workspace](#) button.
5. Upload your files to your own heroku directory on cloud and watch your slack bot in action.
6. View Your Directories: Use this tab to switch between your directories and upload your bot to that directory

View Your Directory

5) Specifically Programmed JavaScript Environment



JavaScript Editor

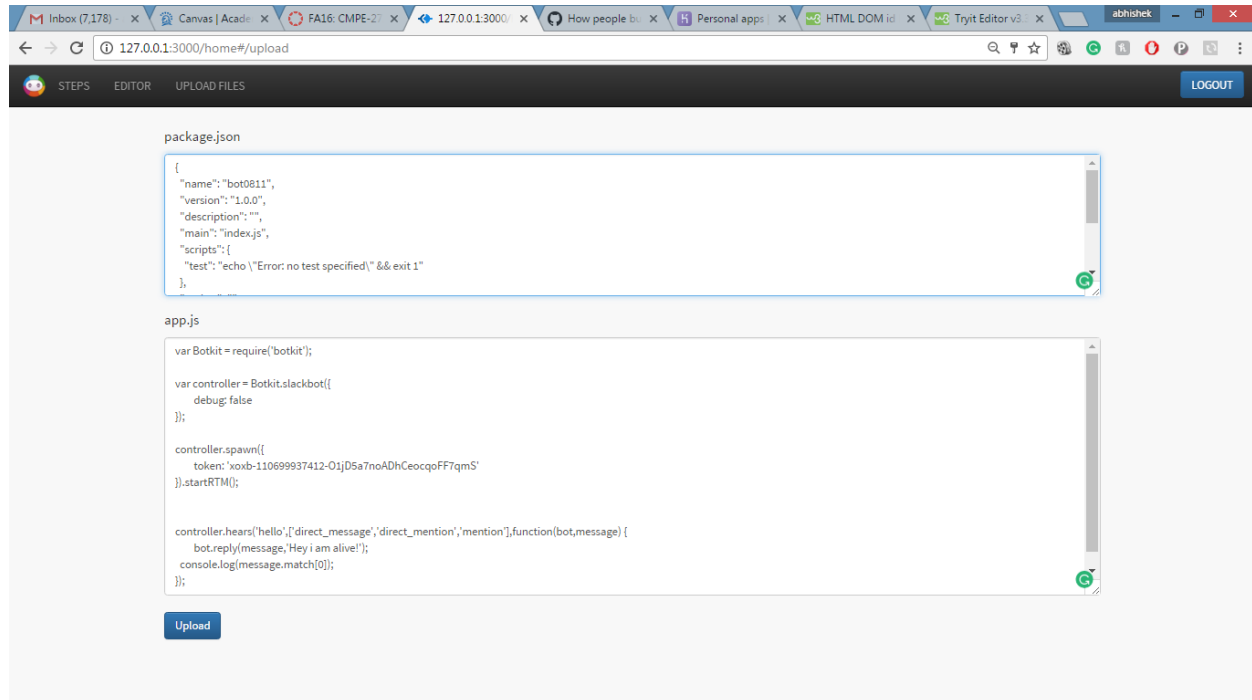
Type your code below

```

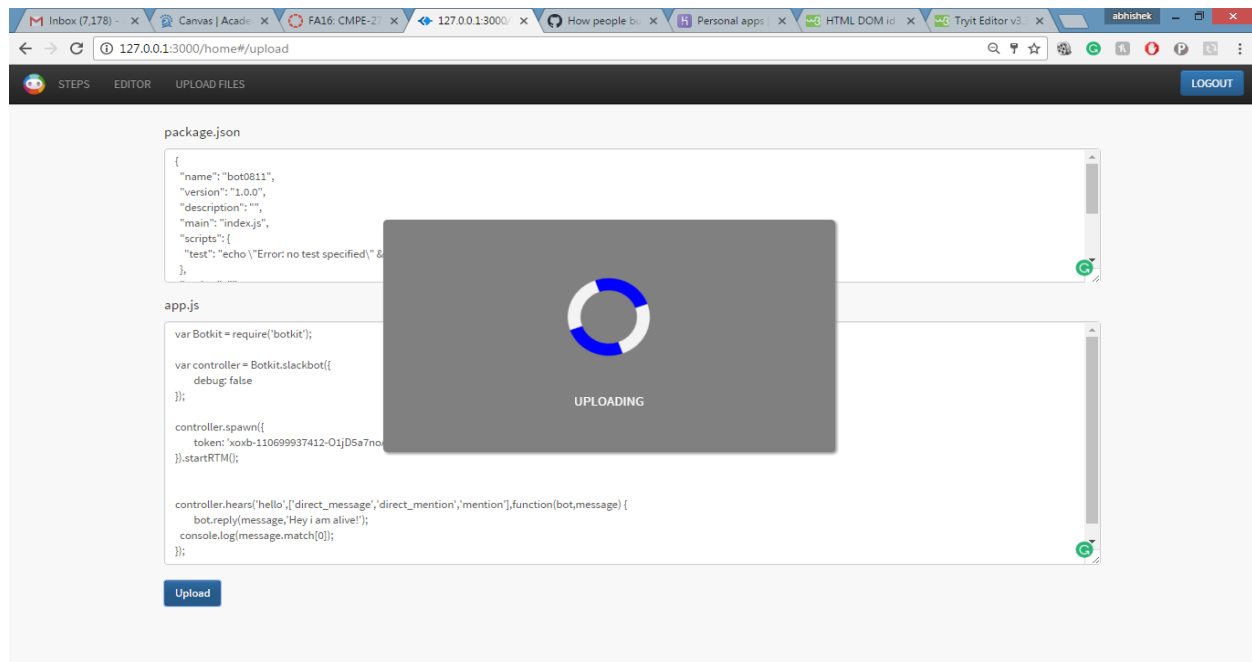
1 function myFunction() {
2   var x = document.getElementById("myAnchor").id;
3   document.getElementById("demo").innerHTML = x;
4 }
    
```

Save workspace

6) Upload to Cloud interface



7) The waiting symbol while we do everything in the backend for you



USECASES:

Our framework can be used to create intelligent bots that could be useful in the following scenarios-

- **Scenario One**

An office work group manager has to maintain list of the things that need to be bought for the meeting. This can be done by adding an inventory management bot to the slack channel. This bot could act as a manager and record the requests for stationary products. While discussing on the channel, if any employee thinks of something to be bought then instead of writing down in a separate notepad and then saving it somewhere, he can just request it over the channel and the bot could record them. This will reduce the overhead of physically recording things or creating a portal where the employee has to raise a request for the same.

- **Scenario Two**

In a project team, while discussing the things sometimes someone uses words which are unknown to the others, in that case a project team can have a slack bot created through BotMan which will search those words in a dictionary and return the results to the team member. This way the communication between the team members will be efficient and productive.

- **Scenario Three**

An additional use case is where an organization like: Dominos can create its bot for order creation. Firms can integrate this bot on to their channel and just by a simple chat could be able to place an order.

CHALLENGES AND FEATURES:

1. Where To Start?

A classic hurdle, which we first envisaged, was where to start dilemma? We were not very much accustomed with the Slack having it only used for project discussions in the past. We first searched regarding what a **SlackBot** is and were utterly mesmerized by the concept and the sheer amount of impact it could further have on the programming group aspect.

Before moving to designing and programming for a framework we decided the language for which we were supposed to give support. After deciding nodejs, we created some sample bots that could help us in the process.

We also decided to implement the Agile Methodologies with Kanban Board (waffle.io).

2. HOW TO HOST ON A CLOUD?

The second concern was to decide how we are going to host a bot on the platform of cloud and make it alive. We were aligned with the concepts of **EC2** and **Docker** initially but to make a '**dockerized container**' or '**EC2 ip**' for each and every bot a user will create through our framework did not seem feasible programmatically and traffic wise as well.

Extensive search yielded us the Heroku platform as a service tool and we could breathe a little when we found that the documentation of their APIs was very elaborately written and could easily be used, as they were open source.

3. Using the APIs thorough hooks and making git bash commands

After deciding to use the Heroku APIs we moved on to next hurdle of integrating git commands with Heroku directory commands.

Extensive search showed that git bash could easily be used in companion with batch files. The batch files had well elaborated batch commands that we used along with the APIs and made the combination of them working.

4. Basic JavaScript environment for programmers

After the git batch files set up the next complete independent thing was the development of an online JavaScript environment, which will eliminate a users, need to install a certain IDE tool just for the sake of development. This initial question got

solved very easily with the help of CodeMirror but to make the environment robust and embed the syntax checking logic was a real big problem for us.

Extensive search on the internet gave us some of the following options,

1. **Using JSHint**
2. **Using JSLint**
3. **Using jsfiddle APIs**
4. **ESLint**

We could list following reason for choosing JSHINT over the other two,

1. JSHint is a forked project from JSLint there are many test cases which JSLint fails to capture when it comes to syntax checking.
2. JSLint has been the main linting tool for JavaScript.
3. JSHint was just a new fork of JSLint, but had not yet diverged much from the original as was in the inception stage.
4. Since then, JSLint has remained pretty much static, while JSHint has changed a great deal - it has thrown away many of JSLint's more antagonistic rules, has added a whole load of new rules, and has generally become more flexible.
5. In addition, another tool ESLint is also now available, which is even more flexible and has more rule options.
6. Regarding the ESLint not much information was readily available.
7. Using the jsfiddle API also was a little complex and we thought of going with the popular and internet communities' choice and thus decided and implemented the JSHint syntax checking.

5. LOGIN OF A USER

In our program scope it was mandatory to use a users session information as we ultimately have to map a bot to the particular user that visited our website.

This was also helpful in creating the session log and information regarding a users bot and log in credentials.

We again had two choices here,

1. To implement and write our own login environment, which would include encrypting a password, mapping the password to a user and bringing into scope a database to maintain a register user service. This would also mean a robust and attack proof system maintenance. This was a challenging task nevertheless not very hard as the node js has libraries for crypting and session maintenance.
2. Second choice was, as our targeted audience a little bit familiar with programming, we thought of making use of a node library called **PassportJS**

with the help of which we could implement the login service of GITHUB to login into our website.

This has following benefits,

1. Robust and secure login

2. A basic user information is obtained along with a mapping primary key, which can be attached to the subsequent activity of the user on our platform. Activities like log generation, bot management, Heroku cloud directory arrangement.

3. This decision yielded in reducing the traffic and load on our system.

6. Session Management

A user must be able to maintain his session throughout the logged in timeline and logout functionalities should destroy that session altogether.

7. SAVE WORK

The user's ability to move freely across any of the tabs gave us a challenge of saving the JavaScript environment workspace in the local storage.

8. UPLOAD TO CLOUD AND MAINTAIN LOG OF CASCADING CHAIN EVENTS

A single upload button will,

1. Install all node package.json dependencies on to the cloud through Heroku API

2. Use the head directory as cloud workspace maintained by Heroku API

3. Commit and push changes in the app.js through batch commands.

Maintaining the log of cascading event is to decide the amount of time the waiting symbol should be shown on the screen.

9. MAINTAINING SIMULTANEOUS BOTS ALIVE

The framework would be useless if multiple bots were not be able to get hosted simultaneously.

We for this same reason included DELETE, MAKE HEAD, VIEW APIs as a part of the framework.

A simple HINTS page will give the user complete information of what he should do in order to host two bots is created,

10. EC2 HOSTING OF THE PROJECT

Many changes had to be incorporated before the project could be successfully hosted on EC2 as a service.

11. LOGS AND JMETER TESTING

We maintained the logs of package.json and user login git information with us so that it could be further used to match with the same users activities.

Maintained logs of,

1. User Session
2. Bot Code specifically package.json for further study
3. Information generated through GitHub passport login.

The JMETER testing involved testing the system availability for 10000 users at a time and counting different graphs for analysis.

The graphs are included in the presentation.

FUTURE SCOPE FOR THE PROJECT:

The project could easily be developed further based on the aspects, which could be listed as follows

1. Support for programming languages other than Node js-

While there is a large community of programmers working on JavaScript related frameworks, we cannot overlook other potential programming platforms. Slack allows developing and deploying bots programmed in languages other than Node, like python and swift. It provides packages which could be easily integrated with these languages in addition to the Node packages. In the future releases of the framework, a good modification to the framework would be to provide support for these additional programming languages as well. This would help us to popularize the framework among a larger pool of slack-bot enthusiasts.

2. Record of the bots created by a particular user date wise -

The framework can be modified with an additional provision of storing the code base of the bots developed by a particular user. When a person logs out from the application, the code base for the Bot is lost. In the next version of the framework, we can provide the user with a cloud-based storage repository option where he could save his work. This would enable him to refer to his past work and also maintain it from time to time.

3. Giving additional controls on the user dashboard-

The current cloud repository controls available to a user are limited to creating, updating and deleting the remote repositories. The dashboard does not provide any control to the user for managing the life cycle of the Bot. Once a bot is made live, the user cannot deactivate the bot locally from the application. A good upgrade to the framework design would be to provide these controls so that the application can become a one-stop solution for developing the bot and managing its life cycle as well.

CONCLUSION:

While bots have long lived in the quieter corners of the Internet, apps like Slack (and WhatsApp, Kik, WeChat and Tinder) are pushing them into the mainstream. A framework like BotMan would give an impetus to the growth and would find its foothold in the market, assisting people to create customized bots for their personal and commercial use.

REFERENCES:

1. <https://github.com/SJSU272Lab/Fall16-Team30/tree/master/BotMan>
2. <https://waffle.io/SJSU272Lab/Fall16-Team30>
3. <https://medium.com/@surmenok/chatbot-architecture-496f5bf820ed#.s5d7mvv87>
4. <https://www.sitepoint.com/custom-slackbot-with-node/>
5. <https://devcenter.heroku.com/articles>
6. <https://www.stackoverflow.com/>