IBOT

Virtual Assistant Tool

Team iBot

Xiao Liang

Xuzhou Yin

Young Liu

Yun Ma

CSE5911 Capstone Design: Software Applications

Table of content

Setup Instructions

1. User setup

It is easy for users to setup. just click here to add iBot to Skype. And it is done!

2. Development setup

For deploy and development purposes, the setup steps might be a little bot more complicated. Please follow the steps:

- Install Node.js
- Create a LUIS endpoint
    - Go to LUIS, create an account, then create a new app.
    - Select `Import Utterance`, upload `Jira.json` from folder `luis-model`, then click `Train`.
    - Click `Publish`, copy the URL.
- Run the bot locally
    - Change the file `.env.example` to `.env`.
        - Change the `LUIS_MODEL_URL` to the URL you just copied.
    - RECOMMENDED: Use `yarn`.
        - Install `yarn`: Tutorial
        - `cd` into code directory
        - `yarn`
        - Launch the bot with `yarn start`
        - Run unit tests with `yarn test`
        - Rerun the code every time file changes (for development purpose only): `yarn launch`
    - You can also use `npm`.
        - `cd` into code directory
        - `npm install`
        - Launch the bot with `npm start`
        - Run unit tests with `npm test`

        - Rerun the code every time file changes (for development purpose only): `npm run launch`
    - Interact with the bot via BotFramework-Emulator
- Deployment
    - Deploy the code to any server, copy down the bot's endpoint.
    - Publish the bot to Microsoft Bot Framework Platform.
    - Config the following environment variable of the bot on the server, then restart it.
        - `MICROSOFT_APP_ID`
        - `MICROSOFT_APP_PASSWORD`
        - `LUIS_MODEL_URL` (Optional if you have already uploaded the `.env` file)
    - You now should be able to access the bot on Skype and any other chat application you fancy.

Introduction

Ibot is designed as a type of virtual assistant that helps gain better user experience when users work. It works as a chatting bot and users can type commands to finish their works. Ibot can record users' needs with a short time in one chat and save them in assigned tools. As a result, it reduces the work loads and saves time for users to improve their work efficiency. To help user with their project management, iBot supports the most frequently used Jira operations such as create cards, move cards, modify cards and assign cards to one of the group members. In addition, user can search cards by typing keywords. One another fancy feature of iBot is that user can even query the burn down rate to track for the current progress of any sprint. To make iBot more secure, user must sign in with valid username, password and Jira URL to do further operations to Jira. If user need to be away for a minute and does not want others to use its account, he or she can sign out temporarily and iBot already remembers the account information of the user. When user gets back and wants to sign in to his or her account, he or she only needs to enter corresponding password. However, if the user wants to switch to another account, he or she will only need to type "SWITCH" command anytime and enter username, password and corresponding Jira URL to connect to sign in to another account.

Problem Statement

Some certain daily tasks employees do everyday require cumbersome steps and unnecessary complexity. For example, this complexity exists in NetJets' IT management with Jira. The workflow is complicated because there are lots of blanks to fill out and there are lots of options to choose from, the user interface is complicated and some of the concepts are confusing. If we can hide those irrelevant details, make it more intuitive for humans and

convert the whole process into conversation, then the unnecessary complexity can be minimized.

IBot aims to solve this problem, it is an intelligent bot which has Natural Language Processing service integrated and interpret the questions into commands. In the behind, it interacts with issue & project tracking tools like Jira, CRM tools like Salesforce, and continuous integration tools like Jenkins so that it helps turn inefficient steps into dialogs to increase productivity for all employees.

Requirement

There are several requirements for the project. The most basic requirement is that iBot should be able to interact with Jira by basic authentication and do some basic operations such as create issue, search issue, and modify issue. For modify issue, iBot should be able to ask user for some keyword of the card user wants to modify, and then ask the user which field to modify (comment, priority, issue status, assignee etc.). In addition, iBot should also be able to use Natural Language Processing to deal with the input from user.

Use Cases

1. Create Issue
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “create a new issue”. iBot displays a list of project names and the user can select the number for one project. iBot displays a list of issue types including task, bug, epic, sub-task and story. The user can select the number for one type. Then iBot summarizes the title of issue from user typing and confirms issue creation. If yes, iBot returns cards that the user can view in browser.

2. Create Story/Task/Bug
    a. Actors:
       User, iBot

b. Basic Flow:

The user types command “create a new story/task/bug”. iBot displays a list of project names and the user can select the number for one project. Then iBot summarizes the title of story/task/bug from user typing and confirms story/task/bug creation. If yes, iBot returns cards that the user can view in browser.

3. Create Epic
    a. Actors:
       User
    b. Basic Flow:

The user types command “create a new epic”. iBot displays a list of project names and the user can select the number for one project. iBot summarizes the epic title then epic name from user typing and confirms epic creation. If yes, iBot returns cards that the user can view in browser.

4. Create Sub-task
    a. Actors:
       User, iBot
    b. Basic Flow:

The user types command “create”. iBot displays a list of project names and the user can select the number for one project. iBot displays a list of issue types including task, bug, epic, sub-task and story. The user can select the number for sub-task. iBot summarizes the title of sub-task from user typing and asks for keyword of related tasks. The user can select the number of one task. Then iBot confirms sub-task creation. If yes, iBot returns cards that the user can view in browser.

5. Create Direct Story/Task/Bug
    a. Actors:
       User, iBot
    b. Basic Flow:

The user types command “create a new story/task/bug “(name for issue)” ”. iBot displays a list of project names and the user can select the number for one project. Then iBot confirms story/task/bug creation. If yes, iBot returns cards that the user can view in browser.

6. Move to Current Sprint
    a. Actors:
       User, iBot
    b. Basic Flow:

After creation, iBot asks if user wants to move the issue to current sprint. If yes, iBot moves the issue to current sprint and returns a notice.

7. Store in Backlog
    a. Actors:
       User, iBot
    b. Basic Flow:

After creation, iBot asks if user wants to move the issue to current sprint. If no, iBot stores the issue in the backlog and returns a notice.

8. Modify Issue
    a. Actors:
        User, iBot
    b. Basic Flow:
        The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user can type command “help” to see modify field.

9. Add Description
    a. Actors:
        User, iBot
    b. Basic Flow:
        The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user types command “description” and iBot stores new description from user typing and returns a notice.

10. Update Priority
    a. Actors:
        User, iBot
    b. Basic Flow:
        The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user types command “priority”. iBot displays a list of priority selection and the user can select the number for one priority. iBot stores the new priority and returns a notice.

11. Add Comment
    a. Actors:
        User, iBot
    b. Basic Flow:
        The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user types command “comment” and iBot stores new description from user typing and returns a notice.

12. Assign to User
    a. Actors:
        User, iBot
    b. Basic Flow:
        The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field

the user wants to modify. The user types command “assign”. iBot displays a list of user names and the user can select the number for one user. iBot stores the new user name and returns a notice.

13. Change Status
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user types command “status”. iBot displays a list of status selection and the user can select the number for one status. iBot stores the new status and returns a notice.

14. Exit Modify
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “modify issue”. iBot displays a list of project names and the user can select the number for one project. iBot asks for keyword of related issue and the user can select the number for one issue. Then iBot asks for the field the user wants to modify. The user types command “Exit”. iBot exits the modify process and returns a notice.

15. Search related issue
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “search”. iBot asks for keyword of related issue and the user can type the keywords. Then iBot returns a horizontal list of cards that user can view in browser.

16. Search unrelated Issue
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “search”. iBot asks for keyword of related issue and the user can type the keywords. If no related issue, iBot returns a failure notice.

17. View issues assigned to users
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “view-assign”. iBot displays a list of user names and the user can select the number for one user. Then iBot displays a list of board names and the user can select the number for one board. iBot returns a horizontal list of cards assigned to that user which can be viewed in browser.

18. Ask for Burn Down Rate
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “burn down rate”. iBot displays a list of project names and the user can select the number for one project. Then iBot displays a list of possible sprint and the user can select the number for one sprint. iBot return a value for burn down rate.

19. Cancel
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “CANCEL” during creation/search. iBot will cancel creation/search process and returns a notice.

20. Exit
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “exit” during modify fields. iBot will cancel modify process and returns a notice.

21. Sign In
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types any commands without sign in. iBot will ask for sign in and the user types command “LOGIN” to log in. iBot asks for username, password and related URL from user. iBot returns a notice and the user can type commands to work.

22. Sign Out
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “LOGOUT” after signing in. iBot will sign out and returns a notice.

23. Switch user
    a. Actors:
       User, iBot
    b. Basic Flow:
       The user types command “SWITCH” after signing in. iBot will ask for new username, password and related URL from user.

Use Case Diagram

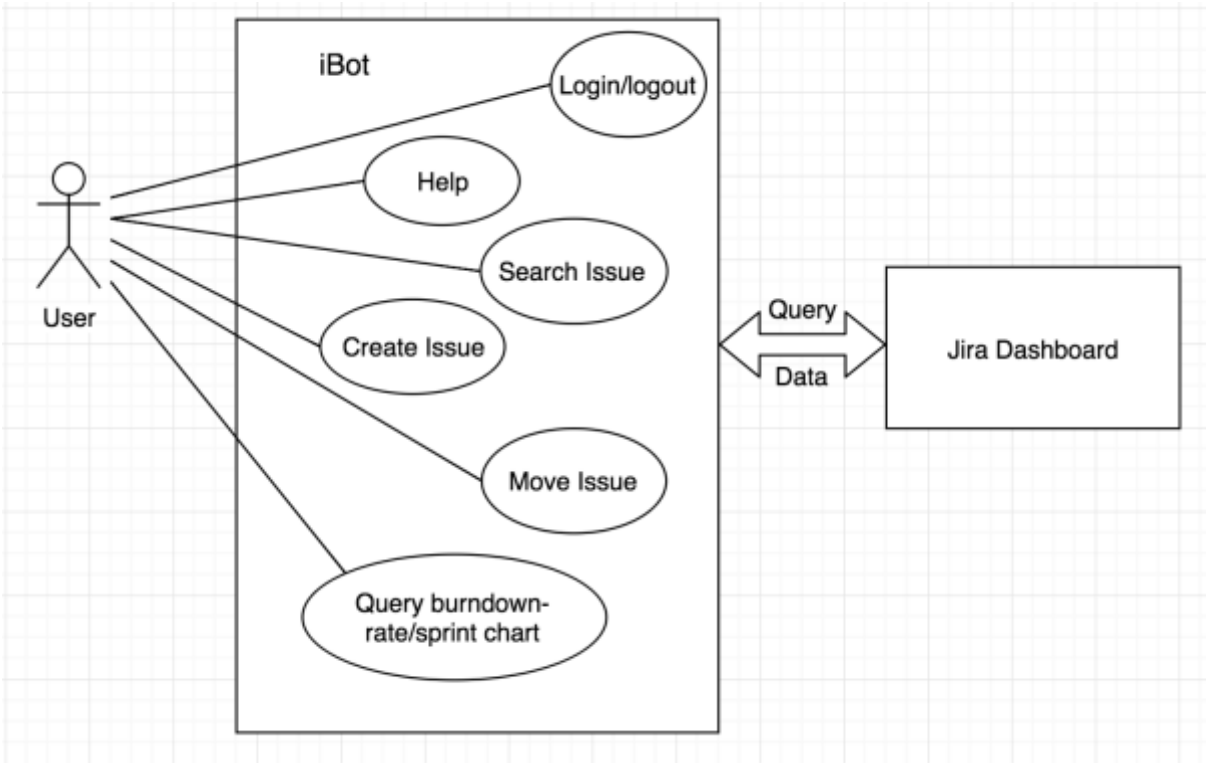

Figure 1: Use Case Diagram

Non-functional requirements

There are several non-functional requirements. The first one is appearance. iBot should have a clean, simple style User Interface so that user feels comfortable when he or she uses it. Since iBot is a chatting assistant, it interacts with users through chatting channel such as Skype and Slack. So there should not be many texts and the texts should be well-formatted. Other than that, iBot should support various platform like Skype, Slack and Facebook Messenger. Another very important requirement is the response time. To impress users and make users feel comfortable, iBot should respond users less than five seconds after it receives inputs. The server iBot is hosting to is also very important. For example, iBot should provide stable services, which means it should be uptime at least 99 percent of time. Next, consider the situation that if users want more features in future, they should be able to develop iBot on the top of current version, which means iBot should have a highly scalable structure that users can scale it up easily. In order to deal with user input and perform high accuracy even if users make some spelling or grammar mistakes. More specifically, iBot ought to get user's desired tasks done within three tries. The last requirement, which is also the most crucial one, is that iBot should not allow unauthorized users to do any tasks and access any informations from Jira.

In Scope

- Interact with Jira:

    - Log in
    - Log out
    - Switch account
    - Create issues
    - Search issues
        - direct search
        - search by user: view issues assigned to specific user
    - Modify issues
        - Add comment
        - Set priority
        - Change issue status
- Use LUIS to interpret natural language.
    - Set up rules to parse users' intent.
- Authentication
    - Cookie based authentication

Out of Scope

- Integrate voice recognition to enable more intuitive interation.
    - Requires extra libraries, accuracies are low.
    - Some applications does not provide direct voice input method.
- Define domain specific language (DSL) to eliminate the ambiguity in natural language.
    - Requires too much time to develop/test.
    - Increases the cost of learning.
- Interact with more DevOps tools
    - Jenkins, Salesforce, BitBucket
- Group chat with multiple users
    - iBot is able to record the important information in the meeting logs
    - Requires multiple authentication

Project Management

The project management includes project schedule, iterations and resource.

Project Schedule

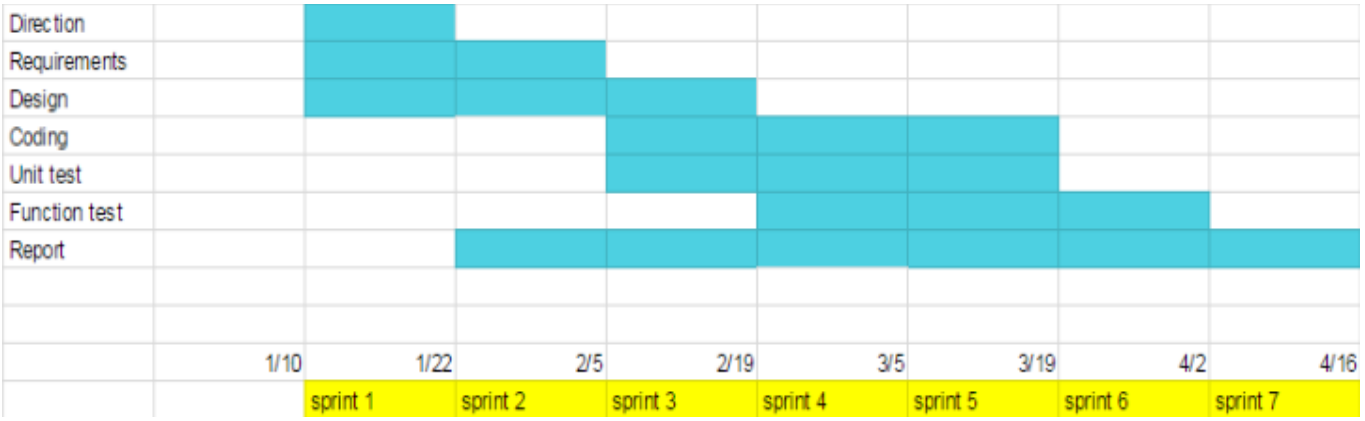

Figure 2: Project Schedule Gantt Chart

Iteration Plan

Iteration One

- Compare Alexa and Hubot: Hubot
- SWOT search on Alexa and Hubot
- Install Hubot on local machines
- Explore APIs of Hubot

Iteration Two

- Turn to Microsoft Bot Framework: Bot
- Bot code base collaborations & continuous integration
- Integrate LUIS into the bot

Iteration Three

- Design new features for iBot
    - Create
    - Modify
    - Ask for burn down rate
- Start to connect to Jira board
- Prepare for Authentication

Iteration Four

- Improve features for iBot
    - Search
    - Add fields for Modify
- Connect to Jira board
- Authentication
- Make LUIS smarter
- Improve the utterance of the bot

Iteration Five

- Improve features for iBot
    - Select fields for modify
    - Search
    - Cancel process
- Search on “fuzzy search”

Iteration Six

- Polish features for iBot
    - Search by user
    - Cancel process
    - Log in/ log out/ switch user
- Deal with multiple boards
- Testing & Debugging

Iteration Seven

- Testing & Debugging
- Arrange resources
    - Project
    - Presentations
    - Reports

Resources

- Open source frameworks: BOT, Node.js and LUIS
- Agile development supported platform: Jira
- Version control & continuous integration: Git & Travis CI
- Microsoft/Jira accounts: APIs
- Azure web service
- Pre-defined LUIS model dataset
- Documentation tools: Google drive
- Team communication tools: Google Hangouts/Wechat
- Personal laptops & IDE: IntelliJ IDEA

Risk

Following are the risks the team encounters for our project:

- Fuzzy search not accurate enough. May need enhance the algorithm
  - Sometimes iBot renders some irrelevant results when user searches issues
- Sign in process is not secure enough
  - Password is not coverred by dots
- LUIS limits the number of times user can interact with iBot
  - One thousand times per month, but can be more if it gets paid
- Sign in page currently only supports Jira account, it will support the accounts from more platforms such as Jenkins and Salesforce

Architecture

Following graph represents the architecture design for iBot. As the graph shows, user interacts with iBot through chatting platforms such as Skype, Slack. The user just need to send his or her desired task to iBot. For example, just a simple text message, “Create a story”. Then iBot brings the messages to LUIS (Language Understanding Intelligent Services). And LUIS uses trained model to extract the intents from the message, and recognizes the task, and output back to iBot. After receiving the task from LUIS, iBot connects to Jira and does the task through Jira. After iBot finishes the task, it returns a successful notification to user says that the desired work has been finished.

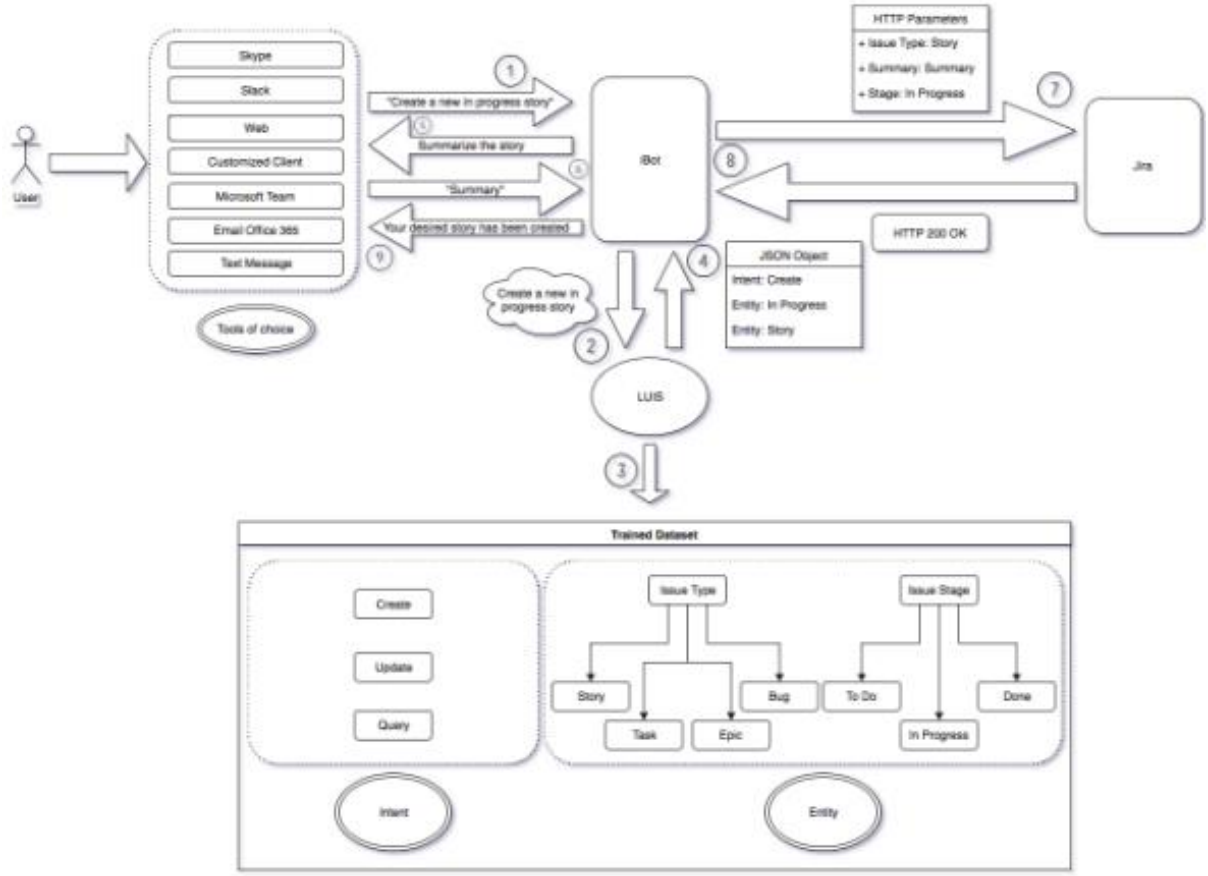

Figure 3: Architecture Design Diagram

Sequence Diagram

The sequence diagram explains the relationships between User, iBot and Tool(Jira). Commands are sent from the user side to iBot side. iBot receives the requests and asks tool to deal with the requests. Tool handles the requests and returns messages back to iBot when finishes. iBot receives messages and replies to user side.
For example, user wants to modify an issue so he/she sends a request to iBot. iBot then asks tool to handle the special area of requests. After the modify process is done, tool returns a confirmation back to iBot and iBot responses to user.

Figure 4: Sequence Diagram

User Interface Diagram

Since iBot is a chatting system, its user interface relies on the chatting platforms. Followings are the user interface of the chatting page on emulator, Skype and Slack:

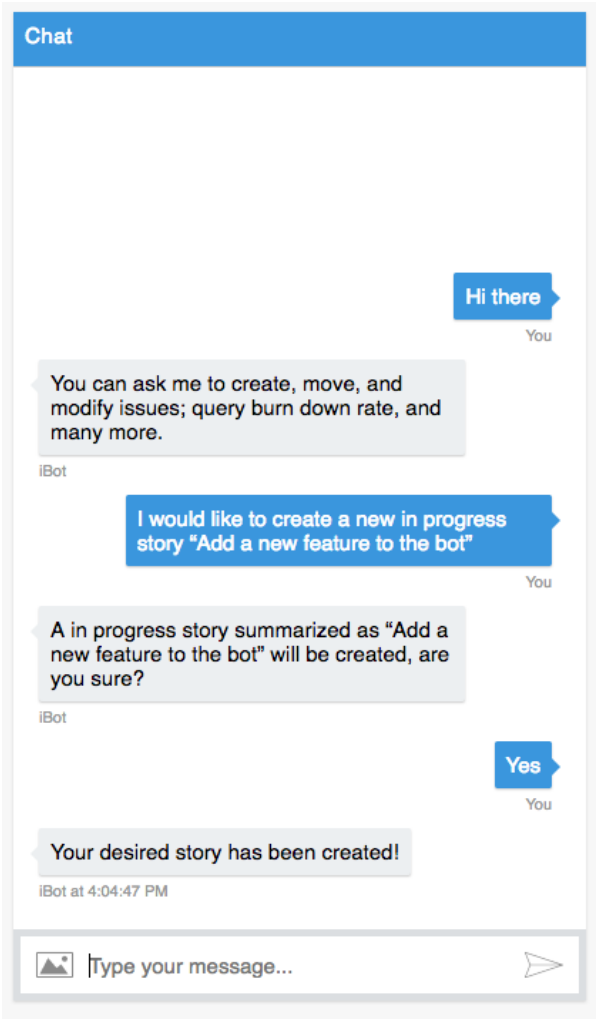

Figure 5: User Interface on Emulator

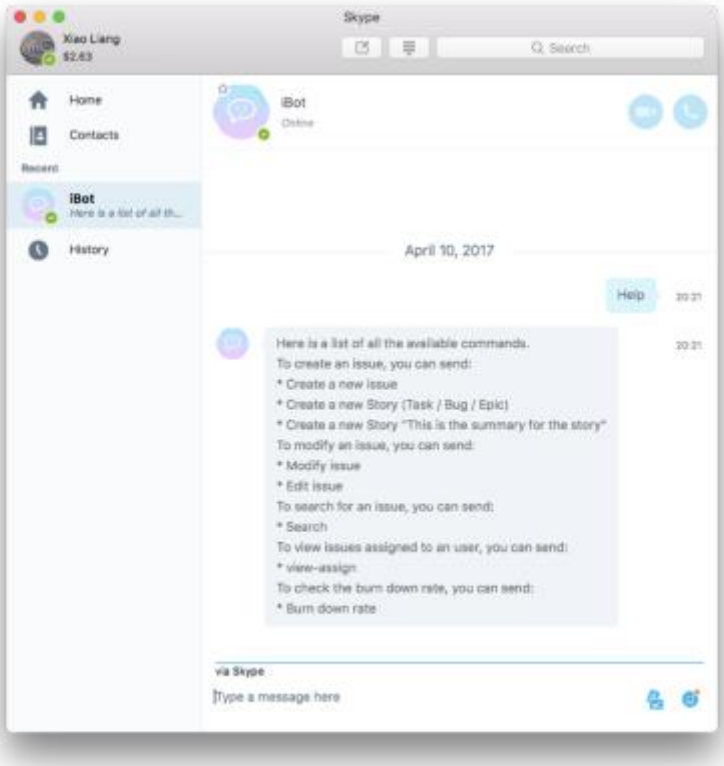

Figure 6: User Interface on Skype

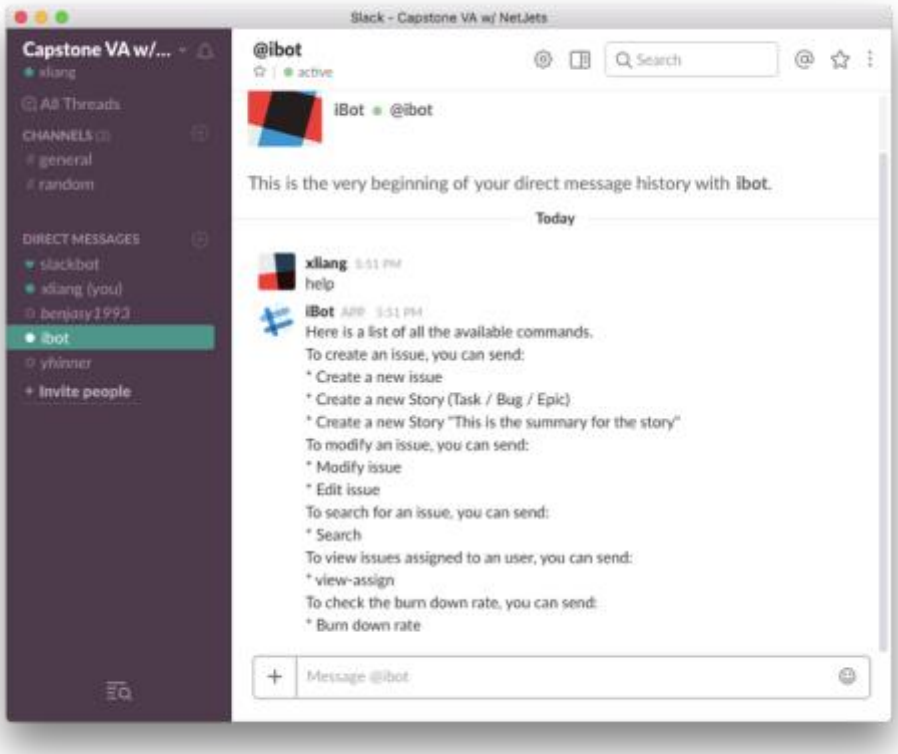

Figure 7: User Interface on Slack

Testing

Following is Use Case Tests

ID	Use Case Name	Testing Done	Result
1	Create a task	Ibot will ask the task details after user types “create a task”	Passed
2	Create an issue	Ibot will ask the issue details after user types “create an issue”	Passed
3	Create a story	Ibot will ask the story details after user types “create a story”	Passed
4	Cancel the process	Ibot will exit after user types “cancel” at any time	Passed
5	Modify priority	Ibot will list the priority selection after user types “change priority”	Passed
6	Modify Status	Ibot will list the status selection after user types “change status”	Passed
7	Modify description	Ibot will ask the description again after user types “change description”	Passed
8	Modify comment	Ibot will ask the comment again after user types “change comment”	Passed
9	Assign to user	Ibot will list the name selection after user types “assign to user”	Passed
12	Search failure	Ibot will return a error message if search fails	Passed

Next Steps

NetJets intents on using our project to build a complete chatting assistant which helps the IT staffs to manage the complete lifecycle of their projects. iBot now support almost all frequently used Jira operations. So, the next big step is to support other DevOps tools such as BitBucket for version control, Jenkins for continuous integration and Salesforce for customer relationship management. At the same time, iBot should be more intelligent. For example, it will support voice input and recognize user desired task from voice messages.

Suggestions for Future Improvements

Although iBot now support almost all frequently used Jira operations, there are still some functionalities that need to be improved:

- integrating more DevOps tools
  - BitBucket
  - Jenkins
  - Salesforce
- Customized features
  - View calendar
  - Check emails
  - Write summary
- Vocie input
  - voice recognition
- Group chat with multiple users
  - iBot takes notes at group meetings