**IBOT**

**Virtual Assistant Tool**

Xiao Liang, Xuzhou Yin, Yun Ma, Young Liu

CSE5911 Capstone Design: Software Applications

**Table of content**

Introduction………………………………………………………………………………………...2

Problem statement………………………………………………………………………………..3

Requirements…………………………………………………………………………………......4

Use cases………………………………………………………………………………...5

User case Diagram……………………………………………………………………….8

Non-functional requirement……………………………………………………………...9

in scope/ out of scope…………………………………………………………………...4

Project Management……………………………………………………………………………...5

Project Schedule………………………………………………………………………….5

Iteration Plan………………………………………………………………………………5

Resources………………….……………………………………………………………...6

Risk………….…..………………………………………………………………………………....7

Architecture………………………………………………………………………………………...8

Sequence diagram………………………………………………………………………..9

User Interface Design…………………………………………………………………..10

Testing………………………………………………………………………………………..……11

**Introduction**

Nowadays, lots of works are finished on the computers through the Internet. Employees check their emails, view personal calendars and summary weekly reports on various types of office software. These repeated works can be dull and boring. Therefore, the demand of some kinds of virtual assistant increases that could help employees simplify work steps and improve work efficiency.

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.

**Problem Statement**

Some certain daily tasks employees do everyday require cumbersome steps and unnecessary complexity.

IBot aims to solve this problem, it is an intelligent bot which has Natural Language Processing service integrated and 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.

**Requirements**

The requirements includes the introduction of features.

*Use Cases*

1. Create Issue
2. Actors:

User, iBot

1. 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.

1. Create Story/Task/Bug
   1. Actors:

User, iBot

* 1. 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.

1. Create Epic
   1. Actors:

User

* 1. 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.

1. Create Sub-task
   1. Actors:

User, iBot

* 1. 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.

1. Create Direct Story/Task/Bug
   1. Actors:

User, iBot

* 1. 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.

1. Move to Current Sprint
   1. Actors:

User, iBot

* 1. 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.

1. Store in Backlog
   1. Actors:

User, iBot

* 1. 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.

1. Modify Issue
   1. Actors:

User, iBot

* 1. 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.

1. Add Description
   1. Actors:

User, iBot

* 1. 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.

1. Update Priority
   1. Actors:

User, iBot

* 1. 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.

1. Add Comment
   1. Actors:

User, iBot

* 1. 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.

1. Assign to User
   1. Actors:

User, iBot

* 1. 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.

1. Change Status
   1. Actors:

User, iBot

* 1. 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.

1. Exit Modify
   1. Actors:

User, iBot

* 1. 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.

1. Search related issue
   1. Actors:

User, iBot

* 1. 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.

1. Search unrelated Issue
2. Actors:

User, iBot

1. 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.

1. View issues assigned to users
2. Actors:

User, iBot

1. 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.

1. Ask for Burn Down Rate
2. Actors:

User, iBot

1. 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.

1. Cancel
2. Actors:

User, iBot

1. Basic Flow:

The user types command “CANCEL” during creation/search. iBot will cancel creation/search process and returns a notice.

1. Exit
2. Actors:

User, iBot

1. Basic Flow:

The user types command “exit” during modify fields. iBot will cancel modify process and returns a notice.

1. Sign In
2. Actors:

User, iBot

1. 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.

1. Sign Out
2. Actors:

User, iBot

1. Basic Flow:

The user types command “LOGOUT” after signing in. iBot will sign out and returns a notice.

1. Switch user
2. Actors:

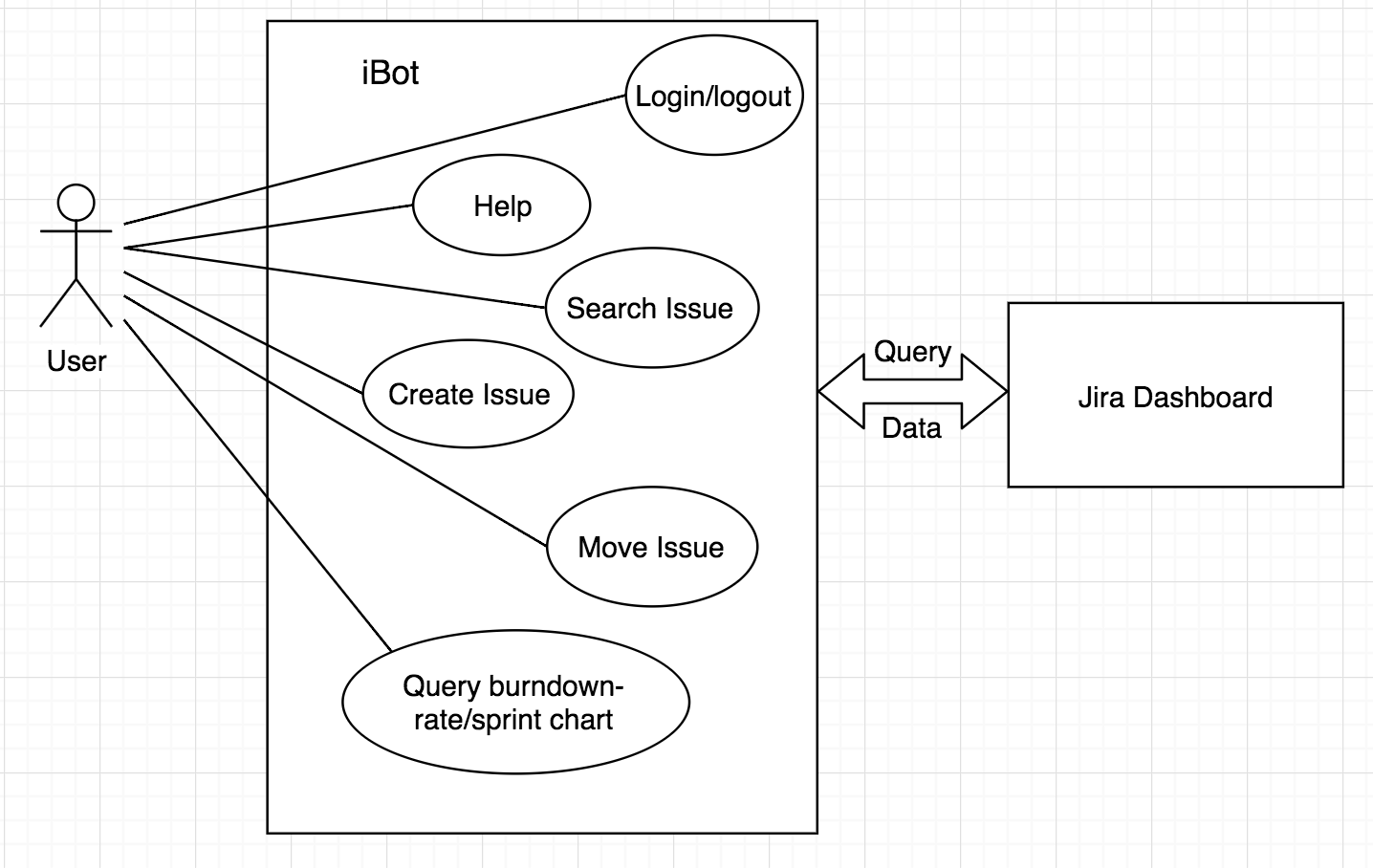
User, iBot

1. 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*

The use case diagram shows the use cases of iBot between User and Jira.



**Non-functional requirements**

//Here is the NFRs.

**In Scope**

**//**here is the in scope

**Out of Scope**

//here is the out of scope

**Project Management**

The project management includes project schedule, iterations and resource.

*Project Schedule*



*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**

//Here is the risks

**Architecture**

The architecture part includes:

* Sequence Diagram
* Architecture Diagram

*Sequence Diagram*

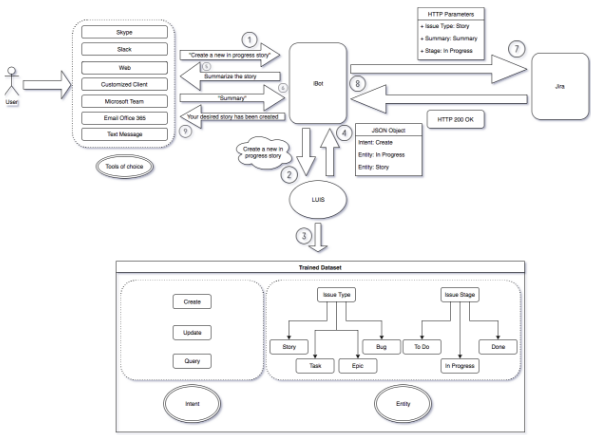
The sequence diagram explains the relationships between User, iBot and Tool(Jira). Commands are sent from the user side to iBot side. iBot receives the requests and asks tool to deal with the requests. Tool handles the requests and returns messages back to iBot when finishes. iBot receives messages and 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.



*Architecture Diagram*

The Architecture Diagram shows the relationships among several sides such as iBot, Jira and LUIS.



**Testing**

**// Manual test**

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 |
| 10 | Search single issue |  | Passed |
| 11 | Search multiple issue |  | Passed |
| 12 | Search failure | Ibot will return a error message if search fails | Passed |