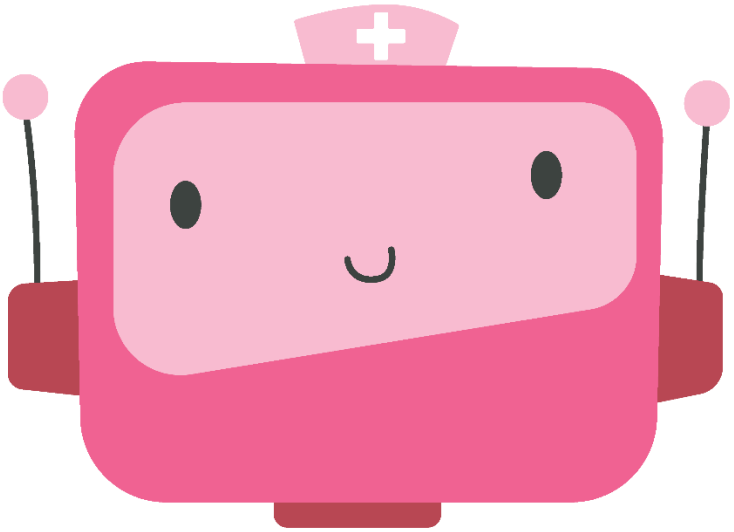
COMP 3613 - Software Engineering II Project

SANOBOT

Healthcare Chatbot



Group Name: **Hotfix 2020**

Group Members:

	Name	*Student Id*	*Email Address*
1.	Fazeeia Mohammed	810001829	mindy.moh@gmail.com
2.	Tyrese Lake	816015110	tlfoot@hotmail.com
3.	Roganci Fontelera	816015552	roganci03@gmail.com

Table of Contents

Table of Figures

Glossary of Terms

Term	Meaning
Chatbot	A computer program designed to simulate a conversation with human users.
Component Testing	Is defined as a software testing type where the testing is performed on individual components.
Database	This is a structure that stores and organizes information
Decision Tree	This is a structure which includes a root node, branches and leaf nodes. Each node is an attribute which defines a test outcome.
DialogflowES	a new technology for creating chatbots.
Dialogue	The flow of interaction between a user and the system
Entity	This is a slot that modifies the user intent.
FLASK Server	This is a micro web framework written in Python
Fulfillment	This is the process of delivering a response to a query.
GitHub	A web-based version control hub
Intent	The goal of the user when typing a question or comment.
POST function	A flask request that may or may not return a response.
SanoBot	The bot developed by Hot-fix.
Sequence Diagram	A sequence diagram is an interaction model between a user and the system.
Symptom	A physical or mental feature which indicates a condition or disease
Unit Testing	Automated test run by software developers to ensure a section of an application meets a specific design.
Use Case	This is a written description that shows how a user will perform a task.
User Story	This is an informal description of one or more features that are written from the perspective of the end user of a system.
Web Hook	This allows intents to be processed from an external environment such as a python server.

Introduction

Problem

The scope statement clearly describes what the project will deliver and outlines generally at a high level all the work required for completing the project

In Trinidad and Tobago there has been a recent increase in the amount of cases of COVID due to the overall lack of caution of its citizens. The medical care system before the outbreak was under significant strain with respect to a lack of beds and insufficient medical staff. To compensate for the potential influx of respiratory distress patients a schedule optimizer based on the symptoms of an individual can be implemented.

Proposed Solution:

A web application that allows users to fill out a survey about a health condition or sickness then locate the most suitable nearby health center/hospital or recommend an online consultation with an appropriate doctor/specialist. The interaction between the user and system will be handled by a chat-bot. The information collected by the chat-bot will mimic an interaction between a nurse or health desk receptionist and the user. The responses will be dynamic and relevant to the context being discussed.

On completion of the chat-bot interaction the system then determines the most suitable action the user may take; the system may recommend a nearby health center/hospital with appropriate staff and facilities and availability of space or recommend an online consultation with a specialist in a particular field. The system will also take into consideration the urgency of the situation and whether or not the condition is potentially contagious. The system will use GPS to locate the nearest suitable health facility and filter users on the criteria of contagious or not contagious.

The system will also prompt the user to make an appointment if they so require.

Project Background

A study done in 2014 publicized by Knoema stated that Trinidad and Tobago has 3 hospital bed units per thousand people. WHO's recommended standard is 5 beds per thousand population. Management of hospital beds and the spreading of contagious diseases in confined places like clinics are a top priority in preventing greater fatality rates exacerbated by the spread of Covid-19. Compounding the need for greater hospital management and scheduling is the potential rise in medical care workers becoming infected and the resurgence of existing infectious diseases predicted by Tobias Brett and Pejman Rohani. The reasons for inadequate bedding, increased infections in medical staff, a current pandemic, and the potential rise for the resurgence of other infectious diseases highlights a need for more efficient health care. Time is usually wasted negotiating with a receptionist to determine an appropriate time and through nurses collecting symptom information. A simple but effective way to decrease wasted time is to optimize the administrative aspect of health care. There are investments being done in this area which goes even as far as to compare existing clinic appointment schedulers.

Project Objectives (High Level)

In the short term we will decrease the time taken for a user to make an appointment. On a grandeur scale the system would potentially be managing contagious outbreaks based on geo-location and symptoms reported.

Functional Requirements

User Requirements

- The user shall be able to speak to a Chatbot to tell it the symptoms they are currently experiencing.
- The user shall be able to state if the one experiencing the symptoms is them or someone else.
- The user shall be able to find out if they have a contagious illness.
- At the end of the conversation the user will have the option of creating an appointment in the selected health center.
- The user shall be able to log in and view their appointments and be able to cancel appointments.

System Requirements

- The system shall be able to determine what illness(es) the user may have.
- The system shall be able to determine whether or not an illness is contagious or not.
- The system should give advice on what to do if the illness is contagious to prevent it from spreading to someone else.
- The system shall create appointments with a nearby health facility.
- The system should be able to determine the best health facility to occupy the user based on the criteria of location, symptoms and doctor's specialty and personal protective gear preparedness.
- The system shall be able to understand a modicum of slang geared towards symptom discernment.

Non-Functional Requirements

1. **Availability:**
   The system shall be available for 24 hours from Monday to Saturday, including holidays. On Sundays the system shall be available for 23 hours and maintenance time shall be from 4:00 am to 5:00 am.

2. **Usability:**
   The program shall have a layout with a monochromatic scheme that is easy to use and understand as well as fit the theme of health. The appearance should also be sensitive to users and should not include too much red or black. The system language shall be English (Trinidad and Tobago).

3. **Capacity:**
   The system shall be able to handle 600 requests per minute.

4. **Compatibility and Portability:**
   The system requires a minimum of 500MB of RAM and 1GB of storage and shall work on most x32 bit and x64 bit operating systems such as Windows, Android, IOS and Blackberry.

5. **Performance:**
   System shall process queries in at maximum 3 second and have a maximum of 2 seconds load time.

6. **Reliability:**
   Failures shall occur 1-3 times a year with a mean time of 3 hours to recover from failure.

7. **Security:**
   The system shall be consistently compliant with ISO/IEC security standards and will protect itself against Trojan attacks, BOTs, Adware, Ransomware, Spyware, Viruses, Worms and any other form of third-party intervention.

Use Case Diagram

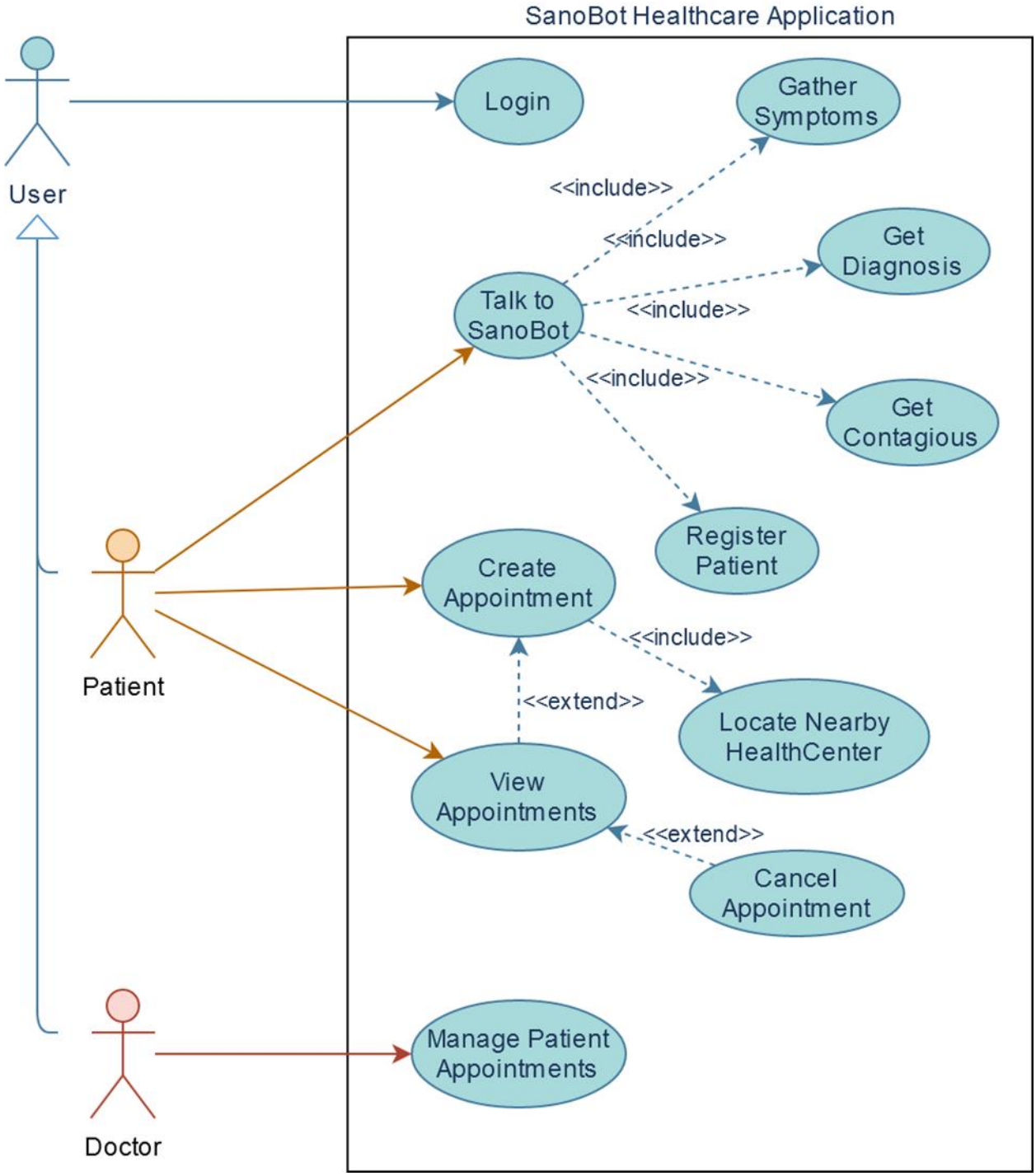


Figure 1: Use-Case Diagram

Users

- Patients
  1. Talk to SanoBot to Receive Diagnosis - Users can talk to SanoBot, outline symptoms and then receive a diagnosis based on the symptoms provided as well as additional information on illnesses that they may have.
  2. Create Appointment at Appropriate Hospital/Health Center - Users may chat with SanoBot to create an appointment after giving SanoBot their symptoms. Users may create an appointment for themselves or someone else.
  3. See Nearby Hospitals/Health Centers - Users can talk to SanoBot and ask about nearby locations or SanoBot will tell the User after giving a diagnosis and creating an appointment.
  4. View/Cancel Appointments - Users can view appointments from the home menu or ask SanoBot about their appointments.
  5. Create Account/Login - Users can create an account and login to save personal information about themselves or other patients and save appointments.
- Doctor
  1. Manage Patients Appointments (Doctor) - Doctors can use the system to view all appointments made to them and move or cancel them where necessary.

Use Cases

- Talk to SanoBot to Receive Diagnosis
- See Nearby Hospitals/Health Centers
- Create Appointment At Appropriate Hospital/Health Center
- View/Cancel Appointments
- Create Account/Login
- Manage Patients Appointments (Doctor)

Ranking Use Cases

Evaluates use cases on a scale of 1-5 against 6 criteria.

1. Significant impact on architectural design.
2. Easy to implement but contains significant functionality.
3. Includes risky, time-critical, or complex functions.
4. Involves significant research or new or risky technology.
5. Includes primary business functions.
6. Will increase revenue or decrease costs.

Use - Case Name	Ranking Criteria, 1 to 5						Total score	Priority	Build Cycle
	1	2	3	4	5	6			
Talk to SanoBot to Receive Diagnosis	4	4	2	4	3	4	**21**	High	1
See Nearby Hospitals/Health Centers	4	4	2	3	3	4	**20**	Medium	2
Create Appointment at Hospital/ Health Center	2	3	4	3	4	4	**20**	High	2
View/Cancel Appointments	3	2	2	2	4	3	**16**	Low	3
Create Account/Login	2	2	2	3	2	2	**13**	Low	3
Manage Patients Appointments (Doctor)	2	3	2	1	1	1	**10**	Low	3

Figure 2: Use-Case Ranking Matrix

User Stories

User Story 1: Receiving a Diagnosis

Mary has a runny nose, a dry cough and a low-grade fever at work. Mary can open up the SanoBot Healthcare Application on her phone and open the SanoBot dialog. SanoBot greets Mary and asks her what's wrong. Mary says she has a dry cough and a low-grade fever and what to know if she has something contagious. SanoBot says "Okay you have a cough and a slight-fever, do you have any other requirements?". Mary tells the bot she also has been experiencing a runny nose for the past 6 hours and says that's it. SanoBot tells Mary that she may have the Cold or Influenza and she should avoid being in close contact with individuals. Mary can then wear her mask and ask to leave work early.

Simple Use Case:

Scenario name: Receiving a Diagnosis

Actor Instances: User/Patient

Precondition: The user has a device to access the internet on and has wi-fi to load the relevant network resource.

Step	Descriptions
1	Selects "Speak to SanoBot"
2	User states the symptoms they are experiencing and other information
3	SanoBot would ask if they are experiencing anything else.
4	The User says "That's all"
5	SanoBot processes the user's symptoms and tells them what illness they may have
6	SanoBot tells the user if it is Contagious, Highly Contagious or Not Contagious
7	SanoBot gives further instructions and the user follows instructions where necessary.

Postcondition: SanoBot asks the user if they would like to make an appointment with a nearby hospital/health center.

User Story 2: Making an Appointment

For the past week George has been feeling very hungry and fatigued. He also has had blurry vision for the past two weeks and has been losing weight dramatically. George would really like to visit the appropriate health center. George can open up the SanoBot Healthcare Application on his tablet and open the SanoBot dialog. George tells SanoBot the symptoms he has been experiencing. SanoBot then replies to George that he is experiencing symptoms that resemble diabetes. SanoBot then prompts George with the message "Would you like to create an appointment at the Specialist in Endocrinology, Diabetes and Internal Medicine Health Center nearby?" with the options Yes or No. George selects yes then types in his information and an appointment is created for next week.

Simple Use Case:

Scenario name: Making an Appointment

Actor Instances: User/Patient

Precondition: The use has received a diagnosis from SanoBot.

Step	Descriptions
1	After making a diagnosis. SanoBot displays the message "Would you like to make an appointment with a nearby Hospital or Health center?" with the options "Yes" and "No"
2	The user says yes or selects the "Yes" option displayed by SanoBot
3	SanoBot displays a sub-form in the chat box with various fields to be filled out to make the appointment.
4	The user fills out the information and then selects or types "I'm done"
5	SanoBot determines the most appropriate Health Center and creates the appointment for the Health Center.
6	SanoBot displays the appointment date to the user and tells them how they can cancel the appointment.

Postcondition: SanoBot returns to the home screen and the appointment is added to the appointments list.

User Story 3: Managing Appointments

Ron was very ill yesterday after partying all night. He used the SanoBot app to make an appointment with a doctor. He opens SanoBot and tells it his Symptoms. Sano bot tells Ron that he may be affected by Mild Food-Poisoning and just in case it gets worse he should make an appointment with the San Juan Health Center on Friday. Ron makes the appointment. The next day Ron is no longer ill and would like to cancel his appointments. Ron can now open the SanoBot dialog and say, "Can I view my appointments?". SanoBot then opens the Appointments Management Menu and Ron can then select cancel appointments.

Simple Use Case:

Scenario name: Managing Appointments

Actor Instances: User/Patient

Precondition: The user had made an appointment recently.

Step	Descriptions
1	After making an appointment the user can either select appointments from the front page or ask SanoBot to view appointments.
2	The appointments manager menu opens.
3	The user views their appointment(s) and selects the one which they want to cancel.
4	The user says "Yes" to the prompt to cancel the appointment.

Postcondition: SanoBot returns to the appointments screen and the appointment is removed to the appointments list.

Sequence Diagram

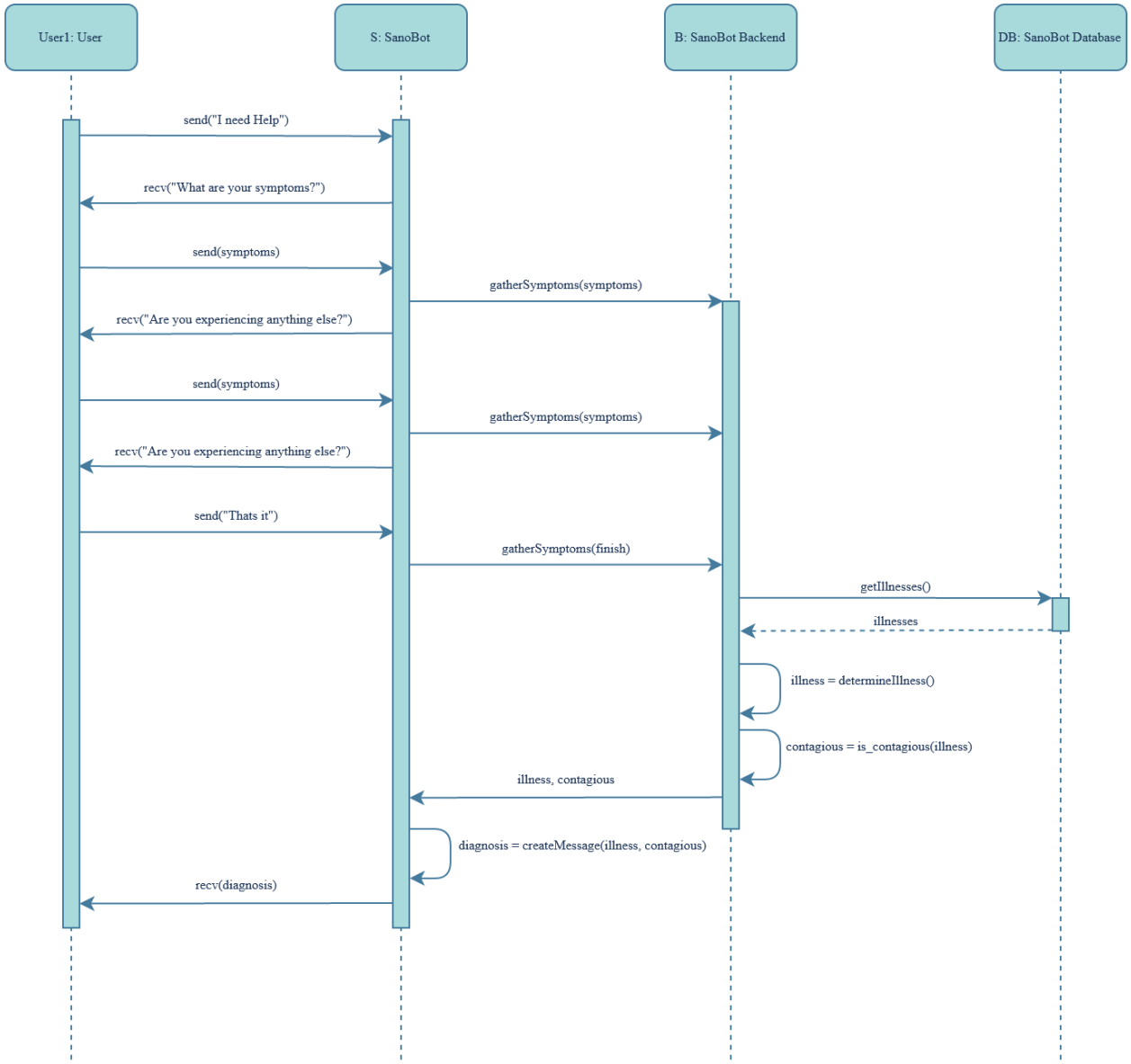


Figure 3: Sequence Diagram

Entity Relationship Diagram

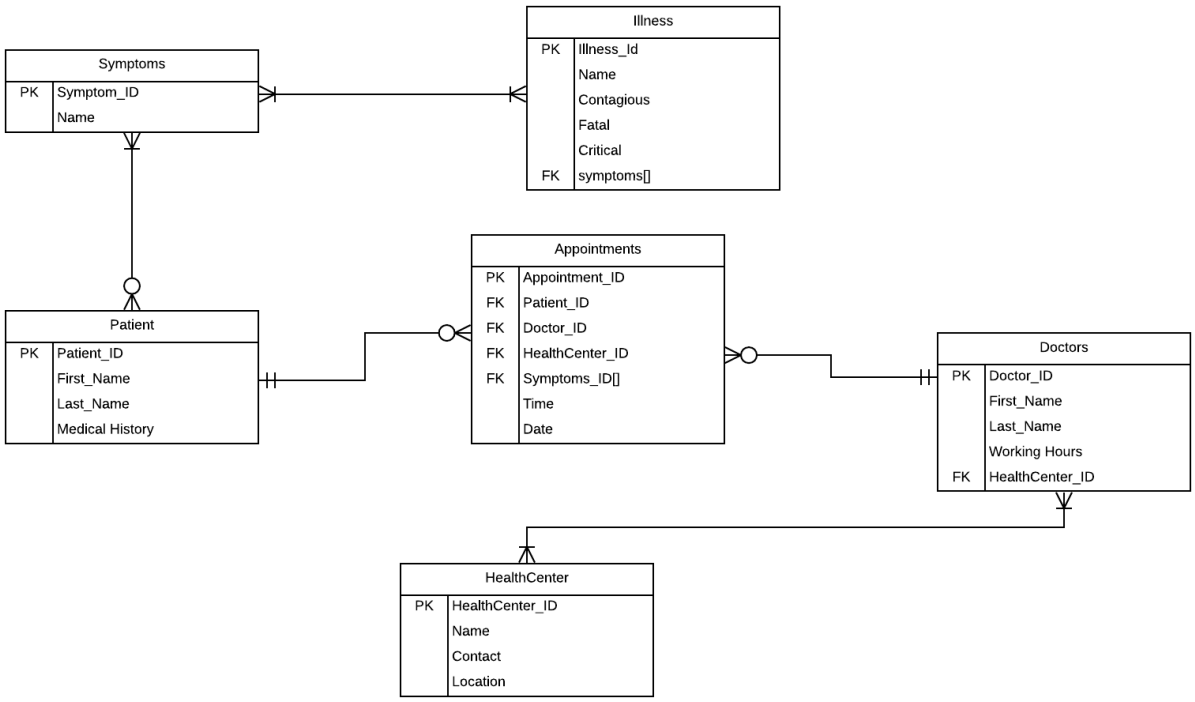


Figure 4: Entity Relationship Diagram

Class Diagram

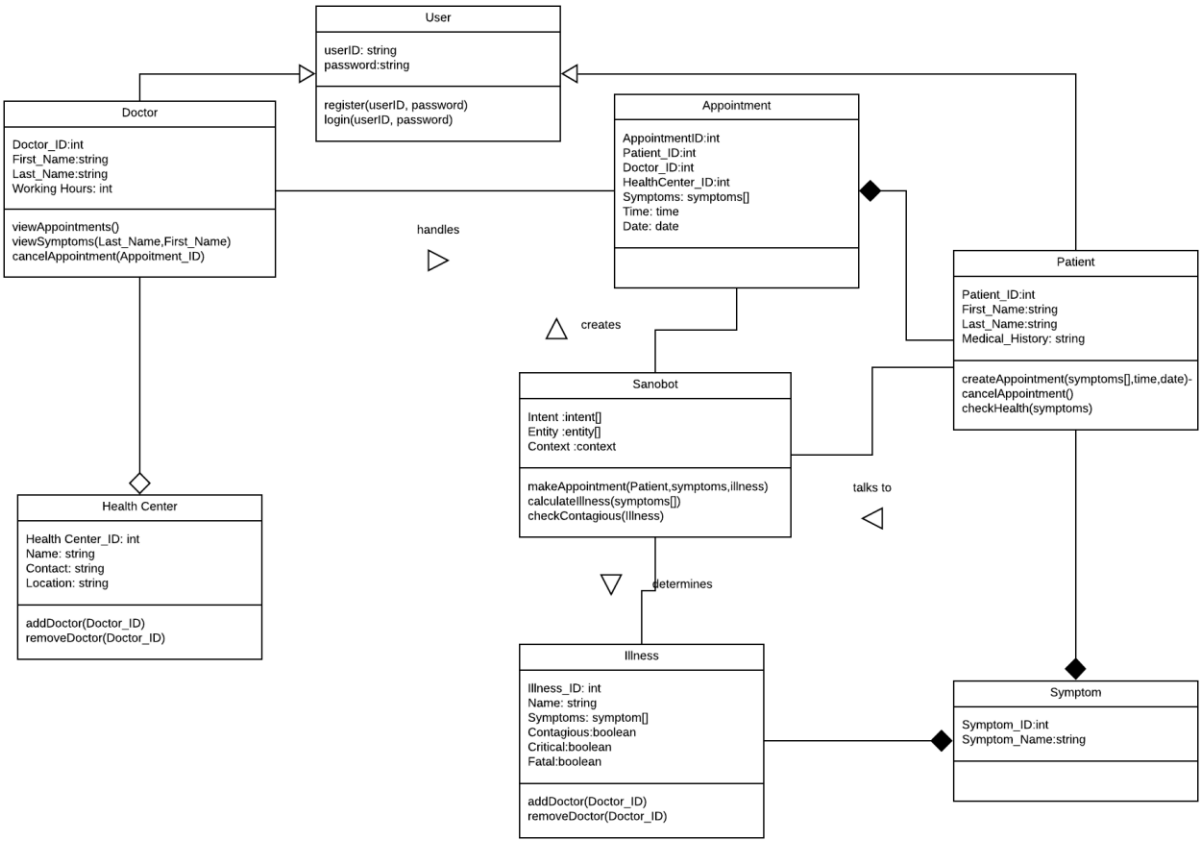


Figure 5: Class Diagram

Architectural Diagram

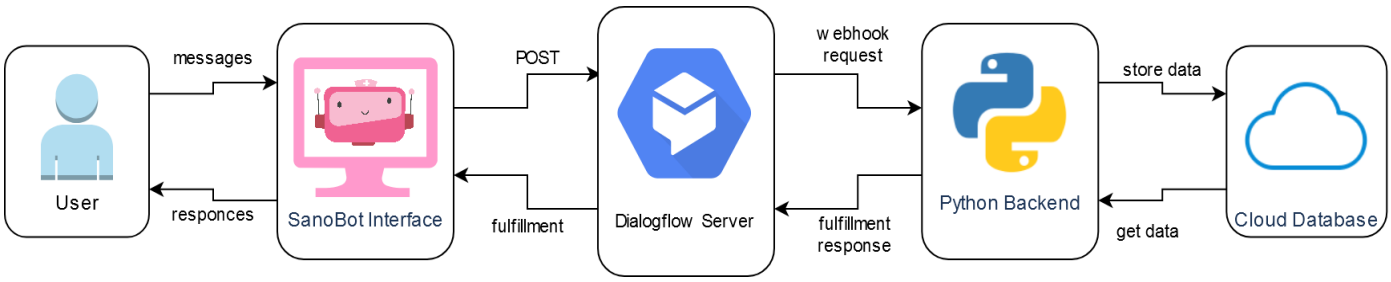


Figure 6: Architectural Diagram

Proof of Methodology

Scrum Methodology

For this project, the methodology utilized was the Scrum methodology. The Scrum methodology allows flexibility and easy management of tasks. Scrum would allow for continuous development both independently and as groups along with ease of coordination between members for activities. With this methodology the Team was able to continuously produce and test components of our Chatbot efficiently. Additionally, it allowed for frequent group meetings to discuss the progress of our project. In the time span of 28th September to 28th November, 4 Sprints were completed and documented.

Tools Used for Scrum

In order to enable the Scrum methodology, several tools were used to facilitate Scrum activities such as Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.

Trello

The team used Trello to facilitate scrum board management. On the scrum board for each Sprint, the product backlog, release backlog, sprint backlog and completed tasks are displayed where tasks were moved along the sprints at daily scrums and during the retrospective and review meetings.

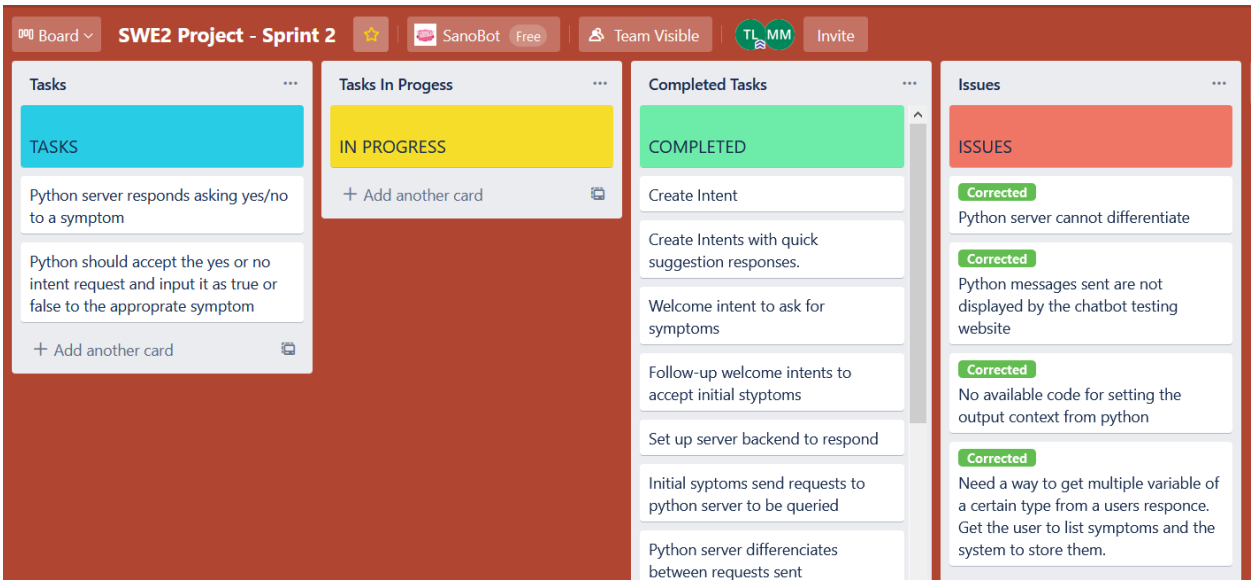


Figure 7: Trello Interface

Microsoft Teams

Microsoft Teams was used by the team to conduct Daily Scrum as well as Sprint Meetings and Retrospective. Microsoft Teams is fitted with group calling, screen sharing, video camera sharing etc. that was used to discuss the Sprint and share ideas and progress.

Microsoft Excel

This spreadsheet software allowed the team to document progress data and collate time taken for each task into Gantt charts and burndown charts for each sprint.

Google Docs

The online document editor allowed the team members to contribute and compile sprint and project documentation into one place.

Sprint 1 Documentation

Time frame

First day: Monday, 28th September 2020

Last day: Wednesday, 21st October 2020

Days worked: 24 Days

Sprint Roles

Product Owner: Fazeeia Mohammed

Scrum Master: Roganci Fontelera

Lead Developer: Tyrese lake

Description

This sprint focused on the proposal, initial documentation and technology research for the SanoBot Application. The proposal included a description of the project, why there is a need for it, the project's scope, the technologies intended to be used and the project objectives. The documentation that would be produced at this stage would be: Use Cases and Scenarios Documentation; High-Level Design Documentation; and Architectural Documentation. Lastly, Dialog Language and Chat-Bot technologies would be researched to gain a better understanding of the possibilities of Chatbots and aid the design of the Chatbot.

Gantt Chart

Scrum Master : Roganci Fontelera
Lead developer: Tyrese Lake
Product Owner: Fazeeia Mohammed

TASK	ASSIGNED TO	START	END	DAYS
Determine System Goals	Roganci & Tyrese	9/28/20	10/3/20	6
Create User Requirements	Fazeeia	10/4/20	10/6/20	3
Rank Use Cases	Tyrese	10/7/20	10/9/20	3
Create Class Diagram	Roganci	10/10/20	10/11/20	2
Determine System Requirements	Roganci & Tyrese	10/11/20	10/12/20	2
Determine Architecture	Fazeeia	10/12/20	10/13/20	2
Create Simple Use Cases	Tyrese	10/14/20	10/16/20	3
Create Sequence Diagram	Roganci	10/14/20	10/16/20	3
Create Sequence Dialogue	Fazeeia	10/15/20	10/17/20	3
Determine the User Senarios to be used	Tyrese	10/17/20	10/18/20	2
Class Diagram	Roganci	10/18/20	10/20/20	3
Create User Stories	Roganci & Tyrese	10/19/20	10/21/20	3
Design Architecture Diagram	Fazeeia	10/20/20	10/21/20	2
Sequence Diagram	All Members	10/20/20	10/21/20	2

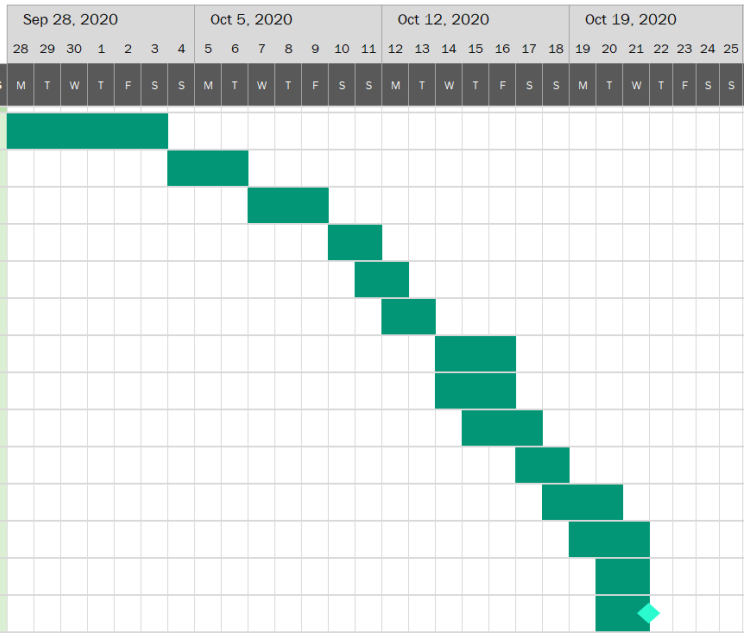


Figure 8: Sprint 1 Gantt Chart

Sprint Backlog

Task No.	Priority	Task	Assignment	Estimated Effort (hours)	Actual Effort (hours)	Status
1	High	Determine System Goals	Roganci & Tyrese	16	18	Completed
2	High	Create User Requirements	Fazeeia	12	9	Completed
3	High	Rank Use Cases	Tyrese	10	9	Completed
4	High	Create Class Diagram	Roganci	6	5	Completed
5	High	Determine System Requirements	Roganci & Tyrese	8	6	Completed
6	High	Determine Architecture	Fazeeia	6	6	Completed
7	High	Create Simple Use Cases	Tyrese	10	9	Completed
8	High	Create Sequence Diagram	Roganci	10	8	Completed
9	High	Create Sequence Dialogue	Fazeeia	10	9	Completed
10	High	Determine the User Scenarios to be used	Tyrese	6	6	Completed
11	High	Class Diagram	Roganci	10	9	Completed
12	High	Create User Stories	Roganci & Tyrese	10	9	Completed
13	High	Design Architecture Diagram	Fazeeia	8	7	Completed
14	High	Sequence Diagram	All Members	6	6	Completed

Figure 9: Sprint 1 Backlog

Burndown Chart

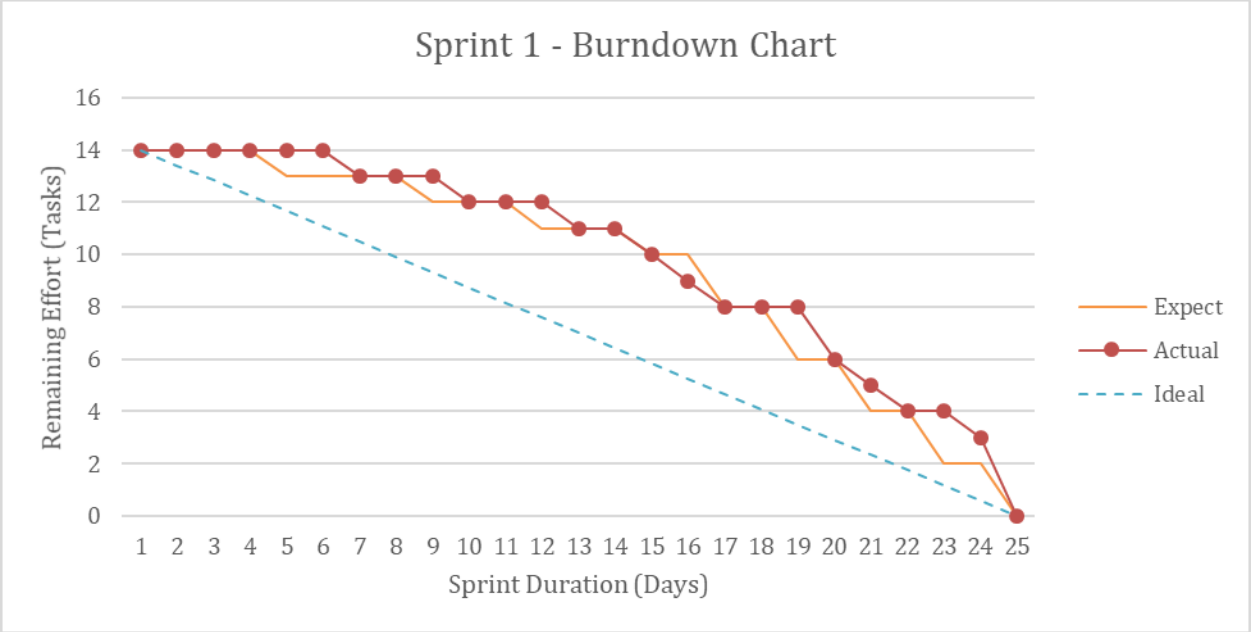


Figure 10: Sprint 1 Burndown Chart

Retrospective

The team had a short retrospective meeting on the 21st of October. This meeting was attended by all team members.

Sprint 1 was overall successful. All outlined tasks were completed within the allotted time frame in an acceptable quality. During this meeting various topics were discussed and the key decisions that were made were:

- The proposal for the chatbot that was made was acceptable however the tasks that are estimated to complete within the allotted time frame may be altered through the course of the next few Sprints.
- The project documentation, particularly the Use Case Documentation and the Project Design Documentation may also change through the course of the next few sprints. This is because general research was done on Dialog Language and Chatbots however not enough research was done to get an exact design of the application.
- The next sprint would focus on doing more research into the Dialog Language and Chatbot technology and actual development of the chatbot would commence.

Sprint 2 Documentation

Time frame

First day: Monday, 26th October 2020

Last day: Wednesday, 6th November 2020

Days worked: 14 Days

Sprint Roles

Product Owner: Fazeeia Mohammed

Scrum Master: Tyrese lake

Lead Developer: Roganci Fontelera

Description

This sprint focused on further research into the selected Dialog Technology, Dialogflow, as well as the creation of basic Chatbot functionality. Dialogflow research would include researching intents, webhook technologies, intent contexts and entities. In this sprint, basic intents would be created to facilitate basic Chatbot communication. Lastly, this sprint would focus on researching and creating a Python webhook back-end for the Dialogflow Chatbot that would facilitate complex and flexible decision making.

Gantt Chart

Scrum Master : Tyrese Lake

Lead developer: Roganci Fontelera

Product Owner: Fazeeia Mohammed

TASK	ASSIGNED TO	START	END	DAYS
Research and Create Intents	Fazeeia	10/26/20	10/26/20	1
Research and Implement Quick Suggestions	Tyrese	10/26/20	10/27/20	2
Create Welcome Intents	Tyrese, Roganci	10/27/20	10/27/20	1
Research and Add Follow-Up Intents	All Members	10/26/20	10/28/20	3
Research Dialog Webhook Backends	All Members	10/26/20	10/28/20	3
Set-Up Dialog Python Server	All Members	10/27/20	10/29/20	3
Modify Server to Detects Different Intents	Tyrese, Roganci	10/27/20	10/29/20	3
Research Python Server Setting Contexts	All Members	10/29/20	10/29/20	1
Program Server to Send Quick Suggestions	All Members	10/30/20	10/30/20	1
Program Server to Set Contexts For Intents	All Members	10/29/20	11/1/20	4
Research Inbound Contexts	All Members	11/1/20	11/1/20	1
Implement Yes/No Questions From Server	Tyrese	11/1/20	11/2/20	2
Research Entities	Fazeeia, Roganci	11/4/20	11/5/20	2
Create a Symtom Entity with Synonyms	All Members	11/4/20	11/6/20	3

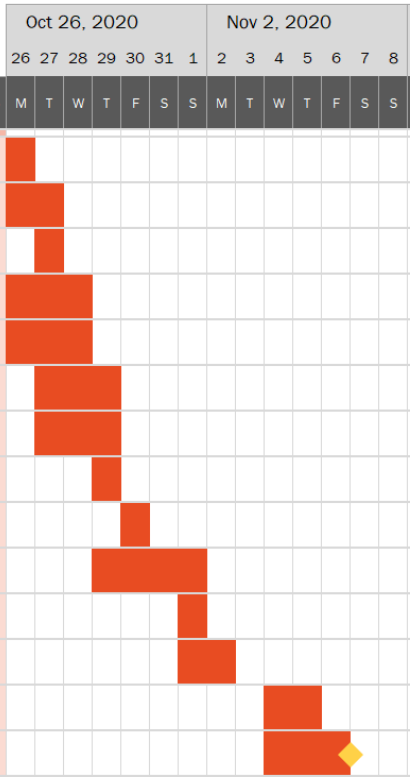


Figure 11: Sprint 2 Gantt Chart

Sprint Backlog

Task No.	Priority	Task	Assignment	Estimated Effort (hours)	Actual Effort (hours)	Status
1	High	Research and Create Intents	Fazeeia	5	6	Completed
2	High	Research and Implement Quick Suggestions	Tyrese	5	6	Completed
3	High	Create Welcome Intents	Tyrese, Roganci	5	4	Completed
4	High	Research and Add Follow-Up Intents	All Members	8	10	Completed
5	High	Research Dialog Webhook Backends	All Members	8	10	Completed
6	High	Set-Up Dialog Python Server	All Members	10	9	Completed
7	High	Modify Server to Detects Different Intents	Tyrese, Roganci	10	9	Completed
8	High	Research Python Server Setting Contexts	All Members	5	4	Completed
9	Medium	Program Server to Send Quick Suggestions	All Members	5	4	Completed
10	Medium	Program Server to Set Contexts for Intents	All Members	10	12	Completed
11	Medium	Research Inbound Contexts	All Members	5	4	Completed
12	Low	Implement Yes/No Questions from Server	Tyrese	5	4	Completed
13	Low	Research Entities	Fazeeia, Roganci	5	4	Completed
14	Low	Create a Symptom Entity with Synonyms	All Members	10	8	Completed

Figure 12: Sprint 2 Backlog

Burndown Chart

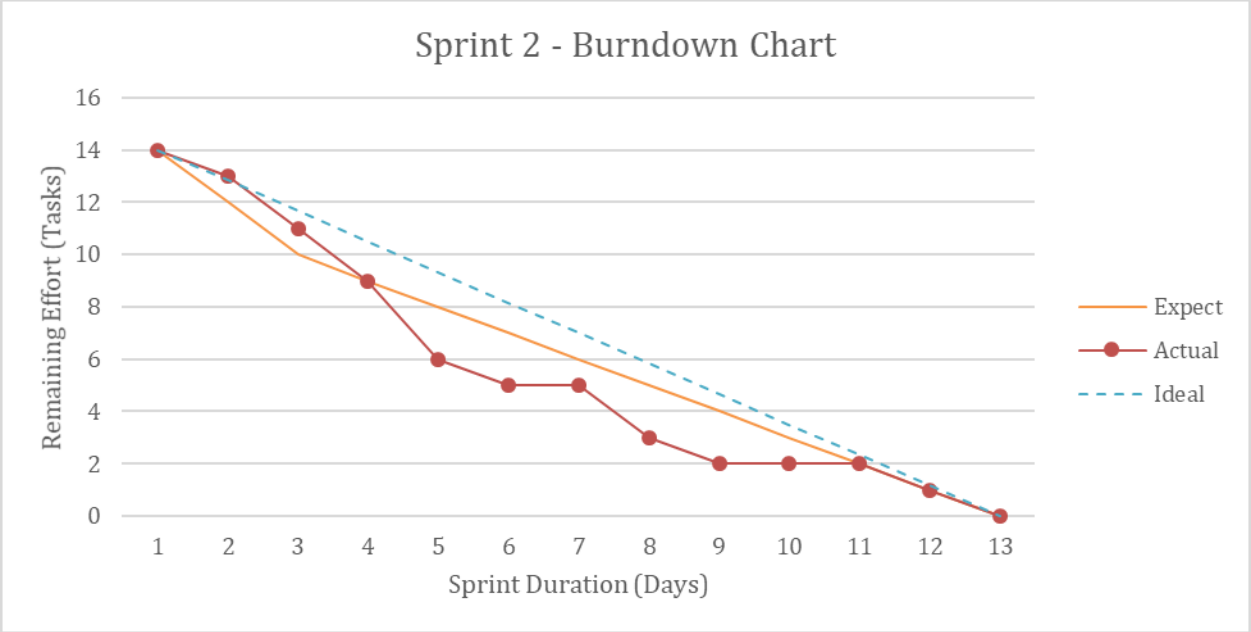


Figure 13: Sprint 2 Burndown Chart

Retrospective

The team had a retrospective meeting on the 6th of November. This meeting was attended by all team members.

Sprint 2 was overall very successful. All outlined tasks were completed within the allotted time frame in an excellent quality. During this meeting various topics were discussed and the key decisions that were made were:

- The chatbot would be called SanoBot.
- Dialogflow ES would be used as opposed to Dialogflow CX as there is a greater availability of resources to learn from.
- The intents, responses and entities created at this phase would be replaced/updated in a later sprint to suit the dataset being utilized and to facilitate more fluent communication.
- The functionality that was researched for Dialog Flow would help in the creation of SanoBot.
- Some of the functionality that was researched and experimented with, such as Quick Suggestions, would not be utilized immediately, however, was still worth researching as it would be used in the future.
- Although there is little available code to allow the communication between Dialogflow and the Python backend, this technology would continue to be used as the team is well versed in python programming and python would allow for very complex decision making later.
- SanoBot would require an interface to communicate with that can communicate with the dialog server.

Sprint 3 Documentation

Time frame

First day: Monday, 9th November 2020

Last day: Monday, 16th November 2020

Days worked: 10 Days

Sprint Roles

Product Owner: Fazeeia Mohammed

Scrum Master: Roganci Fontelera

Lead Developer: Tyrese lake

Description

This sprint focused on the research/creation of a front-end interface for SanoBot as well as connecting this interface to the Dialogflow server for communication. In this Sprint, a prebuilt interface would be researched, selected, modified and used. Alternatively, a front-end would be made from scratch using HTML, CSS and JavaScript. Additionally, a Flask server would be built to render this front-end interface and allow communication with the Dialogflow server using POST functionality. Lastly, this front-end should await, receive and display messages returned from the Dialogflow server.

Gantt Chart

Scrum Master : Roganci Fontelera

Lead developer: Tyrese Lake

Product Owner: Fazeeia Mohammed

TASK	ASSIGNED TO	START	END	DAYS
Rearch Prebuilt Front-End ChatBot Interfaces	Roganci, Fazeeia	11/9/20	11/9/20	1
Create a Front-End Interface	Tyrese, Fazeeia	11/9/20	11/10/20	2
Add Functionality to Front-End Interface	All Members	11/10/20	11/10/20	1
Create Flask Server	All Members	11/11/20	11/12/20	2
Program Flask Server to Host Front-End	All Members	11/12/20	11/14/20	3
Add Basic Routing to Flask Server	All Members	11/12/20	11/14/20	3
Research Python Libaries for Dialogflow	Tyrese	11/13/20	11/13/20	1
Connect Flask Routing to Dialog Server	All Members	11/14/20	11/15/20	2
Update Front-End Send Button to POST to Server	Tyrese, Roganci	11/15/20	11/16/20	2
Update Front-End to Display Received Messages	All Members	11/15/20	11/16/20	2

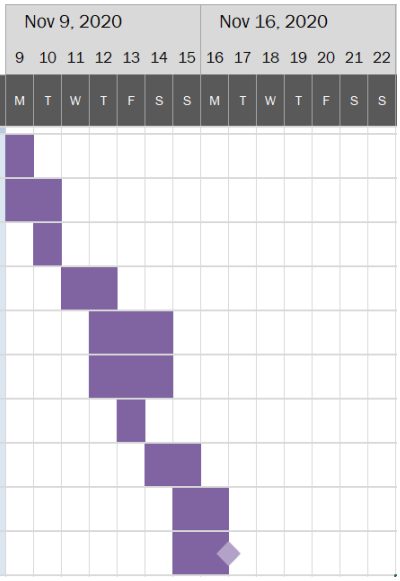


Figure 14: Sprint 3 Gantt Chart

Sprint Backlog

Task No.	Priority	Task	Assignment	Estimated Effort (hours)	Actual Effort (hours)	Status
1	High	Research Prebuilt Front-End Chatbot Interfaces	Roganci, Fazeeia	4	1	Completed
2	High	Create a Front-End Interface	Tyrese, Fazeeia	6	4	Completed
3	High	Add Functionality to Front-End Interface	All Members	4	2	Completed
4	High	Create Flask Server	All Members	6	5	Completed
5	High	Program Flask Server to Host Front-End	All Members	6	6	Completed
6	High	Add Basic Routing to Flask Server	All Members	6	6	Completed
7	High	Research Python Libraries for Dialogflow	Tyrese	4	3	Completed
8	High	Connect Flask Routing to Dialog Server	All Members	10	6	Completed
9	High	Update Front-End Send Button to POST to Server	Tyrese, Roganci	6	6	Completed
10	Medium	Update Front-End to Display Received Messages	All Members	6	6	Completed

Figure 15: Sprint 3 Backlog

Burndown Chart

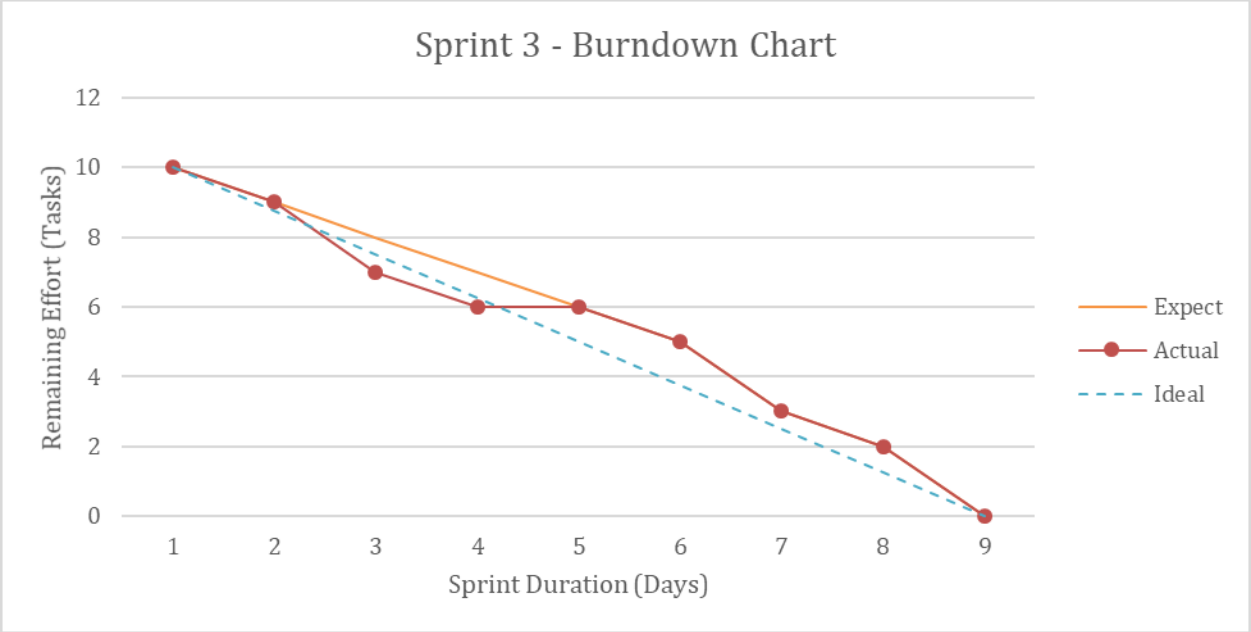


Figure 16: Sprint 3 Burndown Chart

Retrospective

The team had a retrospective meeting on the 16th of November. This meeting was attended by all team members.

Sprint 3 was overall very successful. All outlined tasks were completed within the allotted time frame in an excellent quality. During this meeting various topics were discussed and the key decisions that were made were:

- Pre-built chat interfaces were hard to find and very inflexible, as such the decision to go with a custom-built interface was a good decision as it allowed us to get exactly what we want with easy adjustments.
- The theme and design of the interface perfectly suited SanoBot.
- The interface would require some minor updates later to incorporate asking for names, ending the session and facilitating quick suggestions, however, right now these things were deemed as unnecessary as they have no use currently.
- The next sprint would incorporate aspects of decision making.

Sprint 4 Documentation

Time frame

First day: Monday, 17th November 2020

Last day: Wednesday, 28th November 2020

Days worked: 12 Days

Sprint Roles

Product Owner: Fazeeia Mohammed

Scrum Master: Roganci Fontelera

Lead Developer: Tyrese lake

Description

This sprint focused on implementing the decision-making aspect of the Chatbot. In this Sprint a suitable dataset will be selected and decision-making tools such as decision trees and clusters are researched. These would then be used to design and implement a method that would produce a list of illnesses based on the inputted symptoms. Additionally, the Dialog server would be modified to reflect the test dataset being by incorporating the illnesses and symptoms specified. In this Sprint SanoBot would be made more 'nurse like' by modifying the language used. Lastly, SanoBot would also provide information on what illnesses the user may be describing and provide information on its contagiousness.

Gantt Chart

Scrum Master : Tyrese Lake

Lead developer: Roganci Fontelera

Product Owner: Fazeeia Mohammed

TASK	ASSIGNED TO	START	END	DAYS
Find a Suitable Test Data Set	All Memebers	11/17/20	11/21/20	5
Research Ways of Creating a Decision Tree	All Memebers	11/17/20	11/24/20	8
Research Using Information Gain To Split Data	All Memebers	11/18/20	11/28/20	11
Research Determining Illnesses from Symptoms	Roganci, Fazeeia	11/21/20	11/26/20	6
Develop Code to Determine A List of Illnesses	Roganci	11/21/20	11/26/20	6
Update Symptom Entity	All Memebers	11/21/20	11/25/20	5
Modify Dialogflow to Get All Know Symptoms	All Memebers	11/21/20	11/21/20	1
Update Intents to be "Doctor Like"	Tyrese, Roganci	11/26/20	11/27/20	2
Modify Server to Produce Illnesses Given Symptoms	All Memebers	11/26/20	11/26/20	1
Send User a List of Potential Illnesses	All Memebers	11/26/20	11/27/20	2
Modify Python Server to Determine Contagiousness	Tyrese, Fazeeia	11/27/20	11/27/20	1
Message Sent Should Indicate Contagiousness	Tyrese, Fazeeia	11/27/20	11/28/20	2

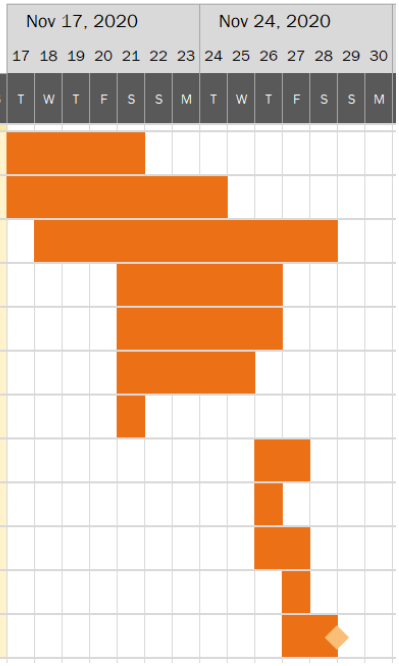


Figure 17: Sprint 4 Gantt Chart

Sprint Backlog

Task No.	Priority	Task	Assignment	Estimated Effort (hours)	Actual Effort (hours)	Status
1	High	Find a Suitable Test Data Set	All Members	10	6	Completed
2	High	Research Ways of Creating a Decision Tree	All Members	10	16	Completed
3	High	Research Using Information Gain to Split Data	All Members	10	16	Not Completed
4	High	Research Determining Illnesses from Symptoms	Roganci, Fazeeia	8	6	Completed
5	High	Develop Code to Determine A List of Illnesses	Roganci	6	6	Completed
6	High	Update Symptom Entity	All Members	6	5	Completed
7	High	Modify Dialogflow to Get All Know Symptoms	All Members	6	2	Completed
8	High	Update Intents to be "Doctor Like"	Tyrese, Roganci	6	2	Completed
9	Medium	Modify Server to Produce Illnesses Given Symptoms	All Members	4	4	Completed
10	Medium	Send User a List of Potential Illnesses	All Members	4	4	Completed
11	Low	Modify Python Server to Determine Contagiousness	Tyrese, Fazeeia	4	2	Completed
12	Low	Message Sent Should Indicate Contagiousness	Tyrese, Fazeeia	4	3	Completed

Figure 18: Sprint 4 Backlog

Burndown Chart

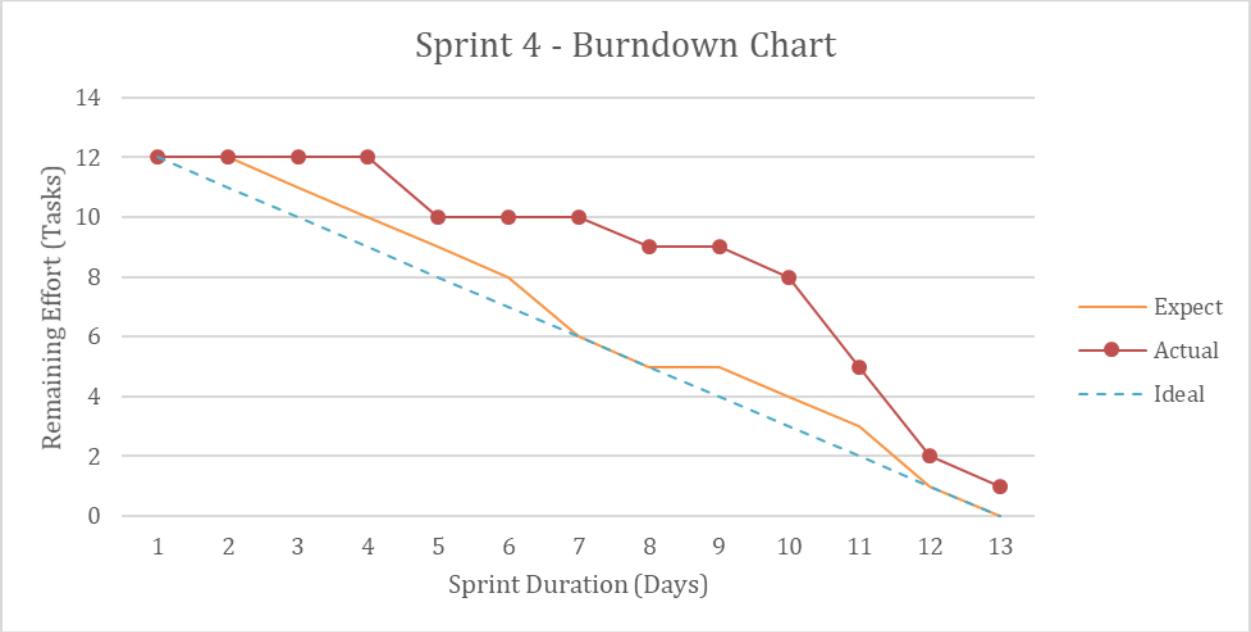


Figure 19: Sprint 4 Burndown Chart

Retrospective

The team had a short retrospective meeting on the 28th of November. This meeting was attended by all team members.

Sprint 4 was satisfactory as there were some unforeseen events that caused the Team to run behind schedule. Almost all outlined tasks were completed within the allotted time frame with the exception of the "Research Using Information Gain To Split Data" task. During this meeting various topics were discussed and the key decisions that were made were:

- The decision tree approach to reducing a large set of illnesses was decided to be left out for this sprint due to the unavailability of a more appropriate test dataset. The project owner decided that this would be done in a later sprint and Sprint 4 would be completed here despite all tasks not being completed.
- The alternative approach of using the gathered symptoms to query a dataset of illnesses and determine a list of potential illnesses worked good enough for this Sprint.
- An information gain approach could have still been used to reduce a large set of illnesses however, the product owner made the decision that it would take too much time to implement for the current sprint.
- SanoBot can be more doctor-like by providing advice as to what to do in the instance that one may have a contagious illness. The team decided that this would need to be done at a later date but for now the state of SanoBot is sufficient.
- This would be the version of SanoBot that would be presented.

Roles

The team consisted of the following:

- PO - Project Owner
- SM - SCRUM Master
- Lead Dev- Lead Developer
- Tester

Product Owner

The role of Product Owner was held by Fazeeia Mohammed BSc Environmental and Biology(minor). She has a background in clinical biochemistry and was the product owner for the duration of the course. Her duties were to update the SCRUM charts regularly, research innovative technologies and adhere to SCRUM protocol. The developmental stages and progression were managed by her. The product owner has a key role to play in liaising with the company to keep the product on track.

SCRUM Master

The role of SCRUM master was alternated over four sprints between Roganci Fontelera and Tyrese Lake. The position of the SCRUM master primary responsibility was to increase the efficiency of the development team. Any issues or impediments were removed by the SCRUM master as part of their responsibilities. They determined the time for the daily SCRUM meetings and facilitated the meeting. They also conducted the retrospective meeting and handled any demonstrations.

Lead Developer

This role was taken up by all three members at varying points. Their main responsibility was writing code and testing features. They viewed the design and implemented it whilst ensuring they function as planned.

Tester

The tester worked in tandem with the developer to test the code previously written.

Scrum Roster

Sprint Number:	1	2	3	4
Tyrese Lake	Lead Dev	SM	Lead Dev	SM
Roganci Fontelera	SM	Lead Dev	SM	Lead Dev
Fazeeia Mohammed	PO	PO	PO	PO

Figure 20: Scrum Roster

Testing Plans - Unit Testing

Unit testing was used to test the functions used in the webhook integration of our Dialog Flow server. The webhook integration was made using python code. The Unit Tests were performed using the python library unit test. Unit tests were done for the following functions:

- getIllnesses( )
- getName( )
- isContagious( )
- getIllnessResponse( )
- removeContext( ) and
- getSymptomsFromContext( ).

Unit Testing was used to ensure that these functions did what they were supposed to do. These unit tests were run each time a major change was made to the python backend.

Unit Tests for Get Illnesses Function

The getIllnesses( ) function returns a list of illnesses given a list of symptoms. The unit tests for this function tests valid inputs, empty inputs and invalid inputs.

Unit Tests

```
#--get illnesses test cases--
def test_get_illness_valid_input(self):
    input = ["sneezing", "chills", "watering from eyes"]
    expect_output = ["Allergy"]
    self.assertEqual(getIllnesses(input), expect_output)
    input = ["itching", "a skin rash"]
    expect_output = ["Fungal Infection", "Chicken Pox"]
    self.assertEqual(getIllnesses(input), expect_output)
    input = ["non_symptom"]
    expect_output = []
    self.assertEqual(getIllnesses(input), expect_output)

def test_get_illness_empty_input(self):
    input = []
    expect_output = ['Fungal Infection', "Allergy", 'Common Cold', 'Pneumonia',
    'Diabetes', 'Chicken Pox', 'Dengue', 'Tuberculosis']
    self.assertEqual(getIllnesses(input), expect_output)

def test_get_illness_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        i = getIllnesses("string")
    with self.assertRaises(ValueError) as E:
        i = getIllnesses(None)
    with self.assertRaises(ValueError) as E:
        i = getIllnesses(10)
```

Figure 21: Get Illnesses Function Unit Tests

Unit Tests for Get Name Function

The getName( ) function returns a sentence appropriate version of an illness name given an illness. For instance, the illness "Allergy" cannot be used by the chatbots dialog as is, it must be converted to "an allergic reaction" by the getName( ) function. The unit tests for this function tests valid inputs and invalid inputs.

Unit Tests

```
#--get name test cases--
def test_get_name_valid_input(self):
    input = "Allergy"
    expect_output = "an allergic reaction"
    self.assertEqual(getName(input), expect_output)
    input = "non_illness"
    expect_output = None
    self.assertEqual(getName(input), expect_output)

def test_get_name_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        i = getName(["list"])
    with self.assertRaises(ValueError) as E:
        i = getName(None)
    with self.assertRaises(ValueError) as E:
        i = getName(10)
```

Figure 22: Get Name Function Unit Tests

Unit Tests for Is Contagious Function

The isContagious( ) function returns the level of contagiousness for a given illness. For instance, the illness "Allergy" is not contagious and would have the contagiousness level of '0', on the other hand, "Tuberculosis" is very contagious and would have a contagiousness level of '2'. The unit tests for this function tests for valid inputs and invalid inputs.

Unit Tests

```
#--get name test cases--
def test_is_contagious_valid_input(self):
    input = "Allergy"
    expect_output = 0
    self.assertEqual(isContagious(input), expect_output)
    input = "non_illness"
    expect_output = None
    self.assertEqual(isContagious(input), expect_output)

def test_is_contagious_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        i = isContagious(["list"])
    with self.assertRaises(ValueError) as E:
        i = isContagious(None)
    with self.assertRaises(ValueError) as E:
        i = isContagious(10)
```

Figure 23: Is Contagious Function Unit Tests

Unit Tests for Get Illnesses Response Function

The getIllnessResponse( ) function returns the appropriate ChatBot response given a list of illnesses. For instance, given the input '["Chicken Pox"]', this function returns:

"You may have chicken pox. This may be very contagious. Avoid being in close proximity to others and contact a doctor immediately. Please be careful."

The unit tests for this function tests valid inputs, invalid illnesses, empty inputs and invalid inputs.

Unit Tests

```
#--get illness response test cases--
def test_get_illness_response_valid_input(self):
    input = ["Fungal Infection", "Chicken Pox"]
    expect_output = "You may have a fungal infection or chicken pox. This may be
    very contagious. Avoid being in close proximity to others and contact a doctor
    immediately. Please be careful."
    self.assertEqual(getIllnessResponse(input), expect_output)
    input = ["Diabetes"]
    expect_output = "You may have diabetes. Please be careful."
    self.assertEqual(getIllnessResponse(input), expect_output)

def test_get_illness_response_invalid_illness(self):
    input = ["non_symptom"]
    with self.assertRaises(TypeError) as E:
        i = getIllnessResponse(input)

def test_get_illness_response_empty_input(self):
    input = []
    expect_output = "Unfortunately, right now I am unable to determine whats wrong.
    We would need to run some tests first to figure out the issue."
    self.assertEqual(getIllnessResponse(input), expect_output)

def test_get_illness_response_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        i = getIllnessResponse("string")
    with self.assertRaises(ValueError) as E:
        i = getIllnessResponse(None)
    with self.assertRaises(ValueError) as E:
        i = getIllnessResponse(10)
```

Figure 24: Get Illnesses Response Function Unit Tests

Unit Tests for Remove Context Function

The removeContext( ) function takes a list of contexts and a context to be removed. For instance, if this function takes a list of contexts "[{"name":"context1"...}, {"name":"context2"...}, {"name":"context3"...}]" and the context "context2, this function would return "[{"name":"context1"...}, {"name":"context3"...}]". The units for this function tests for valid inputs, empty inputs and invalid inputs.

Unit Tests

```
#--remove context test cases--
def test_remove_context_valid_input(self):
    input1 = [{'name': 'session_name/context1', 'value': 1}, {'name':
    'session_name/context2', 'value': 1}, {'name': 'session_name/context3', 'value':
     1}]
    input2 = 'context2'
    expect_output = [{'name': 'session_name/context1', 'value': 1}, {'name':
    'session_name/context3', 'value': 1}]
    self.assertEqual(removeContext(input1, input2), expect_output)
    input1 = [{'name': 'session_name/context1', 'value': 1}, {'name':
    'session_name/context2', 'value': 1}, {'name': 'session_name/context3', 'value':
     1}]
    input2 = 'noncontext'
    expect_output = [{'name': 'session_name/context1', 'value': 1}, {'name':
    'session_name/context2', 'value': 1}, {'name': 'session_name/context3', 'value':
     1}]
    self.assertEqual(removeContext(input1, input2), expect_output)

def test_remove_context_empty_input(self):
    input1 = []
    input2 = 'context'
    expect_output = []
    self.assertEqual(removeContext(input1, input2), expect_output)

def test_remove_context_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        c = removeContext("string", "string")
    with self.assertRaises(ValueError) as E:
        c = removeContext("string", 5)
    with self.assertRaises(ValueError) as E:
        c = removeContext(["non dict"], "string")
```

Figure 25: Remove Context Function Unit Tests

Unit Tests for Get Symptoms from Context Function

The getSymptomsFromContext( ) function takes a list of contexts and returns the list of symptoms from the context with the name "symptomslist". For instance, if this function takes a list of contexts "[{'name': 'session_name/symptomslist',... 'parameters': {'symptomslist' : ['symptom1', 'symptom2']}}]", this function would return "['symptom1', 'symptom2']". The units for this function tests for valid inputs, empty inputs and invalid inputs.

Unit Tests

```
#--get symptoms from context test cases--
def test_get_symptoms_from_context_valid_input(self):
    input1 = [{'name': 'session_name/symptomslist', 'parameters': {"symptomslist" :
    ["symptom1", "symptom2"]}}]
    expect_output = ["symptom1", "symptom2"]
    self.assertEqual(getSymptomsFromContext(input1), expect_output)
    input1 = [{'name': 'session_name/othercontext', 'parameters': {"otherparameter"
    : ["value1", "value2"]}}]
    expect_output = []
    self.assertEqual(getSymptomsFromContext(input1), expect_output)

def test_get_symptoms_from_context_empty_input(self):
    input1 = []
    expect_output = []
    self.assertEqual(getSymptomsFromContext(input1), expect_output)

def test_get_symptoms_from_context_invalid_input(self):
    with self.assertRaises(ValueError) as E:
        c = getSymptomsFromContext("string")
    with self.assertRaises(ValueError) as E:
        c = getSymptomsFromContext(["non dict"])
```

Figure 26: Get Symptoms Function Unit Tests

Testing Plans - Component Testing

Component tested was used to test the dialog component of the Chatbot, this includes the Dialogflow agent and the webhook implementation made with python. The team used component testing to ensure that an entered message received the correct response. Component testing was done by entering a specific message into the Dialogflow Message Terminal given the appropriate inbound context and tests if the correct intent was detected, the correct response and given and the required output contexts were established.

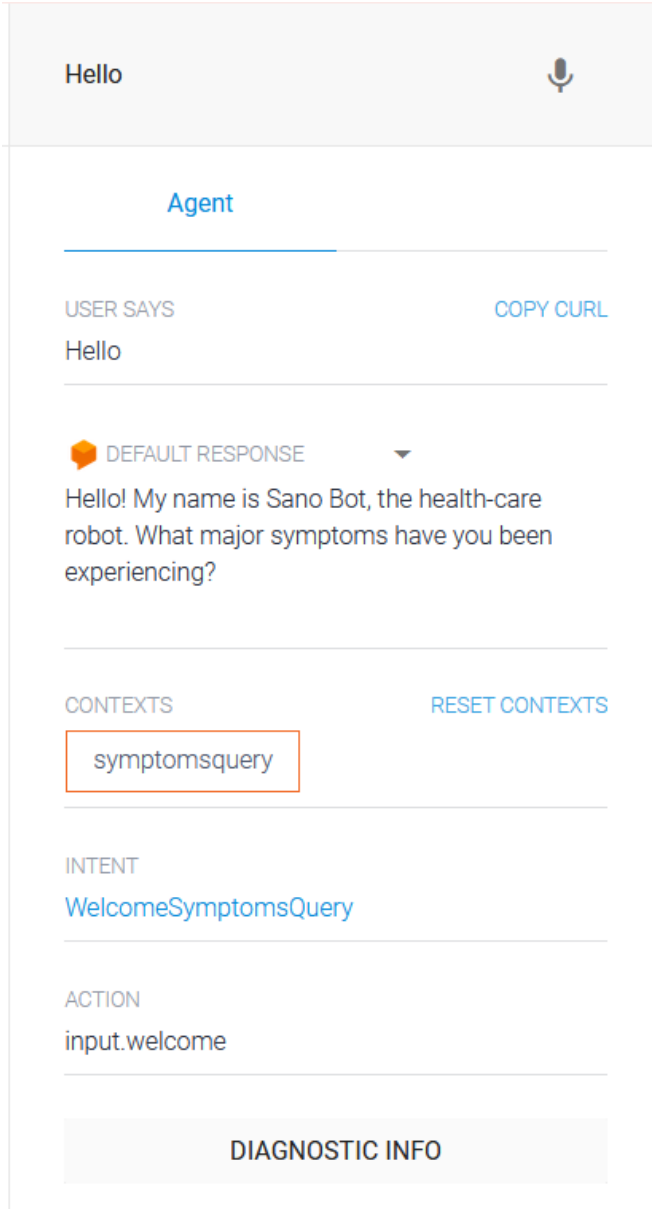


Figure 27: Dialogflow Message Terminal

Initial State

Inbound Context: None

Description: This is the initial state of the Chatbot

Criterion:

- No message was previously sent, or the system finished making a diagnosis

Valid Class:

- User greets the system
- User says they need help

Invalid Class:

- User says a message that the system does not understand.

Boundary:

- N/A

Test Case	Input Type	Message Sent	Detected Intent	Response	Outbound Context
1	Valid	“Hello"	“Welcome Symptoms Query”	“Hello! What major symptoms have you been experiencing?”	“Symptoms Query”
2	Valid	“I need help”	“Welcome Symptoms Query”	“Hello! What major symptoms have you been experiencing?”	“Symptoms Query”
3	Invalid	“Invalid message”	“Default Fallback Intent”	“Can you say that again?”	None

Figure 28: Initial State Component Test

Symptoms Gathering

Inbound Context: "Symptoms Query"

Description: This allows users to list all the symptoms they are experiencing.

Criterion:

- The user has greeted SanoBot or the user has asked SanoBot for help or the user has already given some symptoms.

Valid Class:

- User says one symptom that they are experiencing
- User gives 2 or more symptoms they are experiencing
- User gives more symptoms after already providing symptoms

Invalid Class:

- User says a message that the system does not understand.
- User says "that's it" without listing any symptoms

Boundary:

- Minimum 1 symptom given

Test Case	Input Type	Message Sent	Detected Intent	Response	Outbound Context
1	Valid	"I have a cough"	"Symptoms Gathering"	"So, you are experiencing coughing. Do you have any other symptoms?"	"Symptoms query", "More symptoms query", "symptomslist"
2	Valid	"I am coughing and sneezing"	"Symptoms Gathering"	"So, you are experiencing coughing and sneezing. Do you have any other symptoms?"	"Symptoms query", "More symptoms query", "symptomslist"
2	Valid	"I am also sneezing"	"Symptoms Gathering"	"So, you are experiencing sneezing. Do you have any other symptoms?"	"Symptoms query", "More symptoms query", "symptomslist"
3	Invalid	"Invalid message"	"Default Fallback Intent"	"Can you say that again?"	Unchanged
4	Invalid	"That's all"	"Default Fallback Intent"	"Can you say that again?"	Unchanged

Figure 29: Symptoms Gathering Component Test

Illness Diagnosis

Inbound Context: "More symptoms query"

Description: After user have entered 1 or more symptoms that they are experiencing, they can then receive a diagnosis

Criterion:

- The user has greeted the bot and has entered 1 or more symptoms.

Valid Class:

- User says "that is all" after listing some symptoms that belong to a known illness
- User says "that is all" after listing some symptoms that belong to an unknown illness

Invalid Class:

- User says a message that the system does not understand.

Boundary:

- N/A

Test Case	Input Type	Message Sent	Detected Intent	Response	Outbound Context
1	Valid – Illness Known	"That's all"	"No More Symptoms"	"You may have the common cold. This may be mildly contagious, so avoid too much physical contact with anyone. Please be careful."	Reset
2	Valid – Illness Unknown	"That's all"	"No More Symptoms"	"Unfortunately, right now I am unable to determine what's wrong. We would need to run some tests first to figure out the issue."	Reset
3	Invalid	"Invalid message"	"Default Fallback Intent"	"Can you say that again?"	Unchanged

Figure 30: Illness Diagnosis Component Test

Risk Management

1.	**Risk:** The time required to develop the software is underestimated.		**Strategy:** Investigate buying-in components.	
	Affects:	Project		
	Probability:	High		
	Effects:	Serious	**Strategy Type:**	Avoidance

Figure 31.1: Risk 1

2.	**Risk:** Organizational financial problems force reductions in the project budget		**Strategy:** Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost effective.	
	Affects:	Project		
	Probability:	Low		
	Effects:	Catastrophic	**Strategy Type:**	Minimization

Figure 31.2: Risk 2

3.	**Risk:** It is impossible to recruit staff with the skills required for the project.		**Strategy:** Alert customers to potential difficulties and the possibility of delays; investigate buying-in components.	
	Affects:	Project		
	Probability:	High		
	Effects:	Catastrophic	**Strategy Type:**	Minimization

Figure 31.3: Risk 3

4.	**Risk:** Changes to requirements that require major design rework are proposed.		**Strategy:** Derive traceability information to assess requirements change impact; maximize information hiding in the design.	
	Affects:	Project and product		
	Probability:	Moderate		
	Effects:	Serious	**Strategy Type:**	Minimization

Figure 31.4: Risk 4

5.	**Risk:** The number of users for SanoBot exceeds the limit for Kommunicate Service Package.		**Strategy:** Project the new user population of SanoBot to determine the subscription upgrade required for the long term.	
	Affects:	Product		
	Probability:	Moderate		
	Effects:	Serious	**Strategy Type:**	Avoidance

Figure 31.5: Risk 5

6.	**Risk:** The Dialog Flow becomes discontinued by Google		**Strategy**: Investigate alternative service providers; Notify management that investment into a new technology may be necessary	
	Affects:	Project and project		
	Probability:	Moderate		
	Effects:	Catastrophic	**Strategy Type:**	Minimization

Figure 31.6: Risk 6

7.	**Risk:** The physical database is corrupted or severely damaged due to natural disasters.		Strategy: Have a cloud database which backups software versions and user appointments periodically.	
	Affects:	Product		
	Probability:	Low		
	Effects:	Serious	**Strategy Type:**	Avoidance

Figure 31.7: Risk 7

8.	**Risk:** Key staff are ill at critical times in the project		**Strategy:** Reorganize the team so that there is more overlap of work and people therefore understand each other's jobs.	
	Affects:	Project		
	Probability:	Moderate		
	Effects:	Serious	**Strategy Type:**	Minimization

Figure 31.8: Risk 8

9.	**Risk:** The underlying technology on which the system is built is superseded by new technology.		**Strategy:** Investigate new technology open source to build upon or buy in components .	
	Affects:	Business		
	Probability:	High		
	Effects:	Catastrophic	**Strategy Type:**	Minimization

Figure 31.9: Risk 9

10.	**Risk:** Software tools cannot be integrated		**Strategy:** From the research process, use runner up software tools.	
	Affects:	Project		
	Probability:	High		
	Effects:	Tolerable	**Strategy Type:**	Avoidance

Figure 31.10: Risk 10

Cost Estimation

The cost estimation for SanoBot was modeled using COCOMO II. This algorithmic model is used because its utilization of historical data makes it a reliable and accurate form of estimation.

Software Development (Elaboration and Construction)

Effort	1.7 Person-months
Schedule	4.4 Months
Cost	$17472 USD
Total Equivalent Size	623 SLOC

Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.1	0.6	0.2	$1048
Elaboration	0.4	1.7	0.3	$4193
Construction	1.3	2.8	0.5	$13279
Transition	0.2	0.6	0.4	$2097

Figure 32: Acquisition Phase Distribution Table

Software Effort Distribution

Phase/ Activity	Inception	Elaboration	Construction	Transaction
Management	0.0	0.1	0.1	0.0
Environment/ CM	0.0	0.0	0.1	0.0
Requirements	0.0	0.1	0.1	0.0
Design	0.0	0.2	0.2	0.0
Implementation	0.0	0.1	0.5	0.0
Assessment	0.0	0.0	0.3	0.1
Deployment	0.0	0.0	0.0	0.1

Figure 33: Software Effort Distribution Table

Web Application Layout

Application Layouts were created at the beginning of the design process in order to give the team an idea of how the application would be laid out and where each element would be positioned.

Home Interface Layout

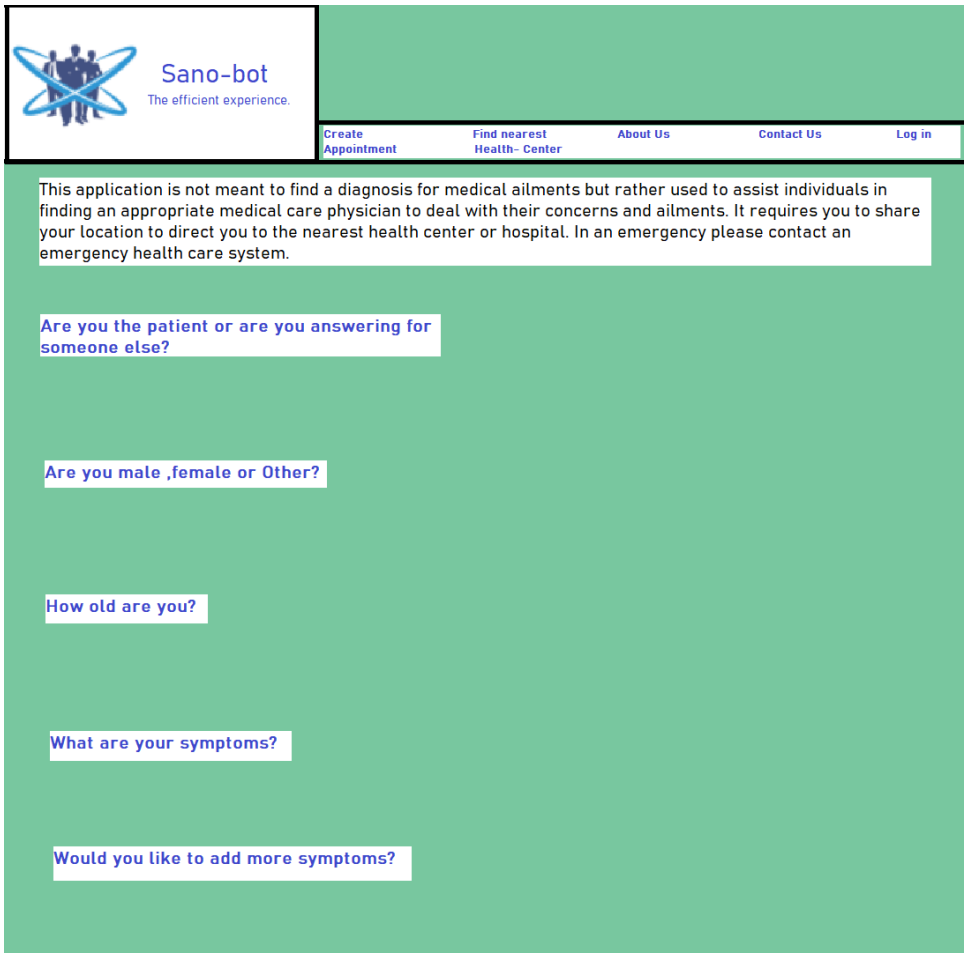


Figure 34: Initial Home Interface Layout

Chatbot Interface Layout - 1

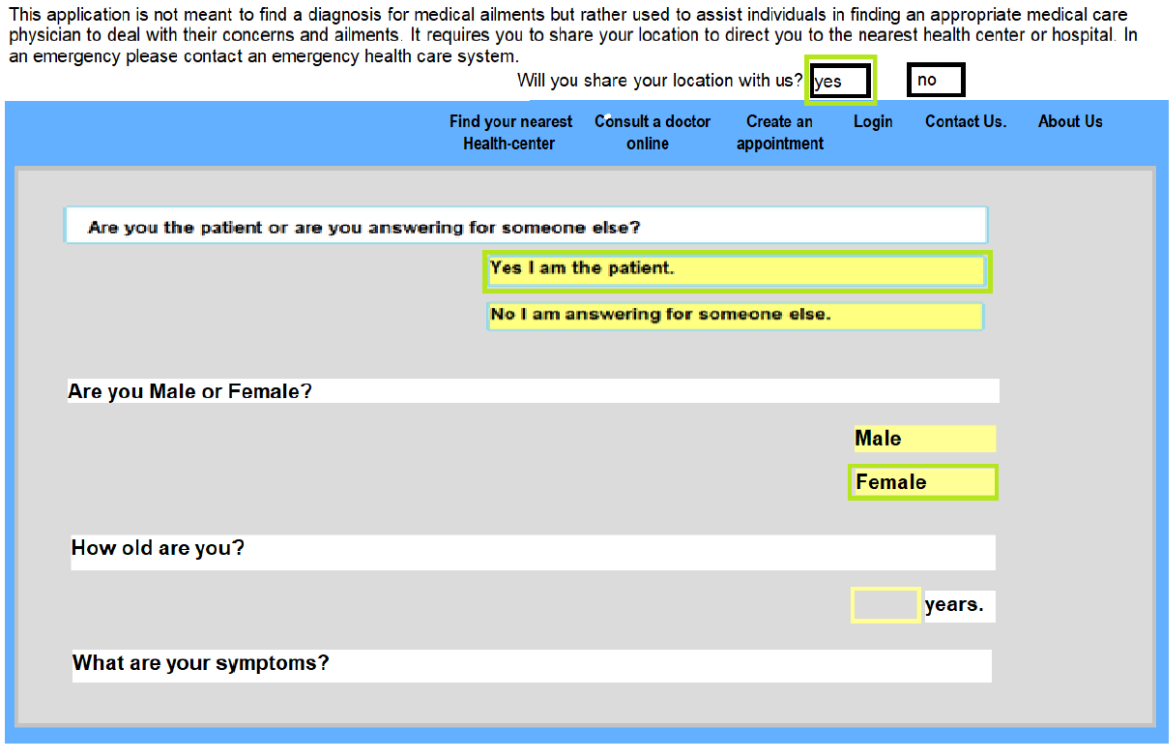


Figure 35: Initial Chatbot Interface Layout

Chatbot Interface Layout - 2

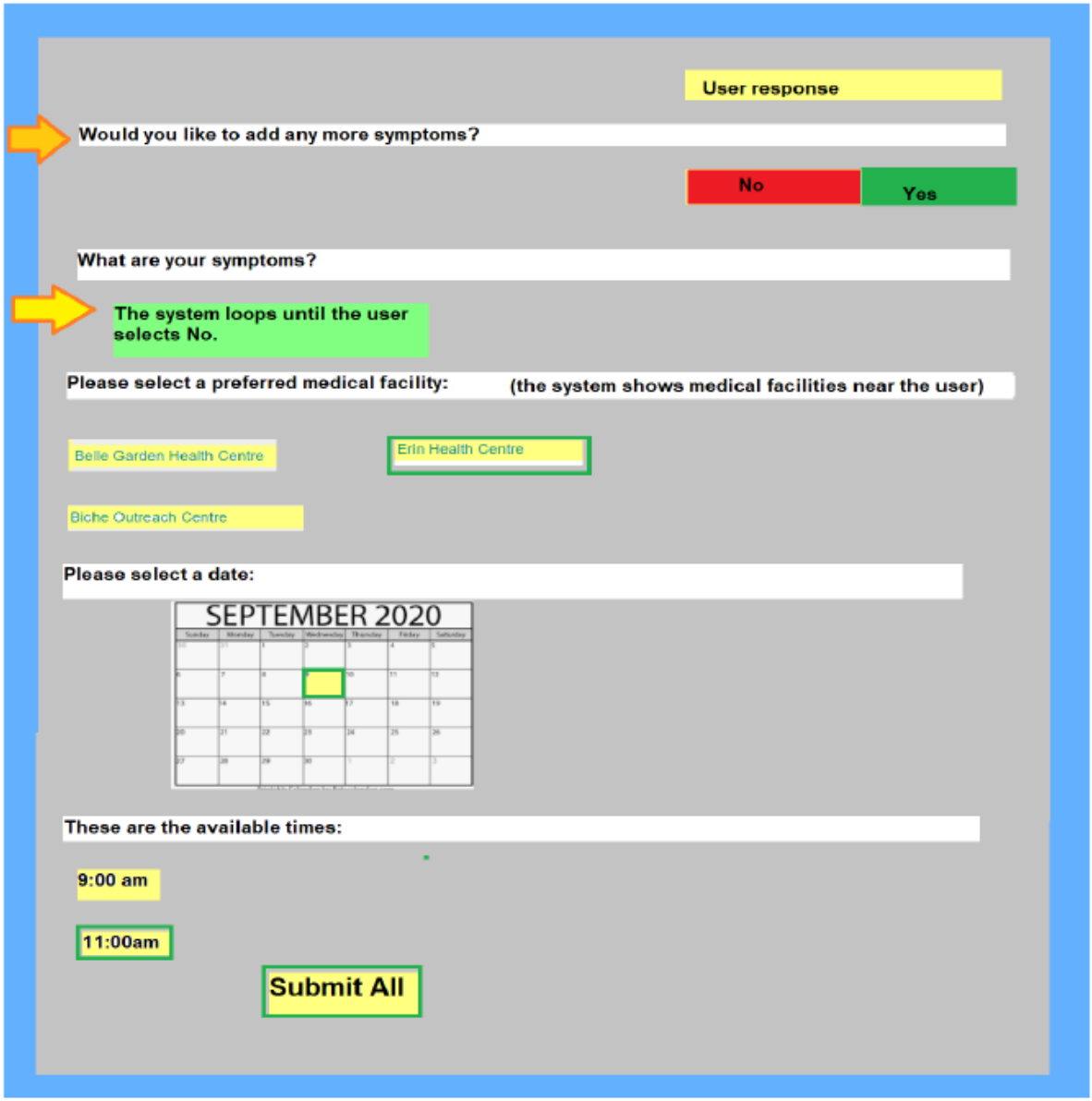


Figure 36: Initial Make Appointment Interface Layout

Working Application

Chatbot Component

The Chatbot Component, SanoBot, was created using: html, CSS, and JavaScript for the front end interface; a flask server to host the webpage and facilitate communication with backend; dialog flow to facilitate text interpretation and response as well as webhook communication and python for the webhook backend.

SanoBot Interface

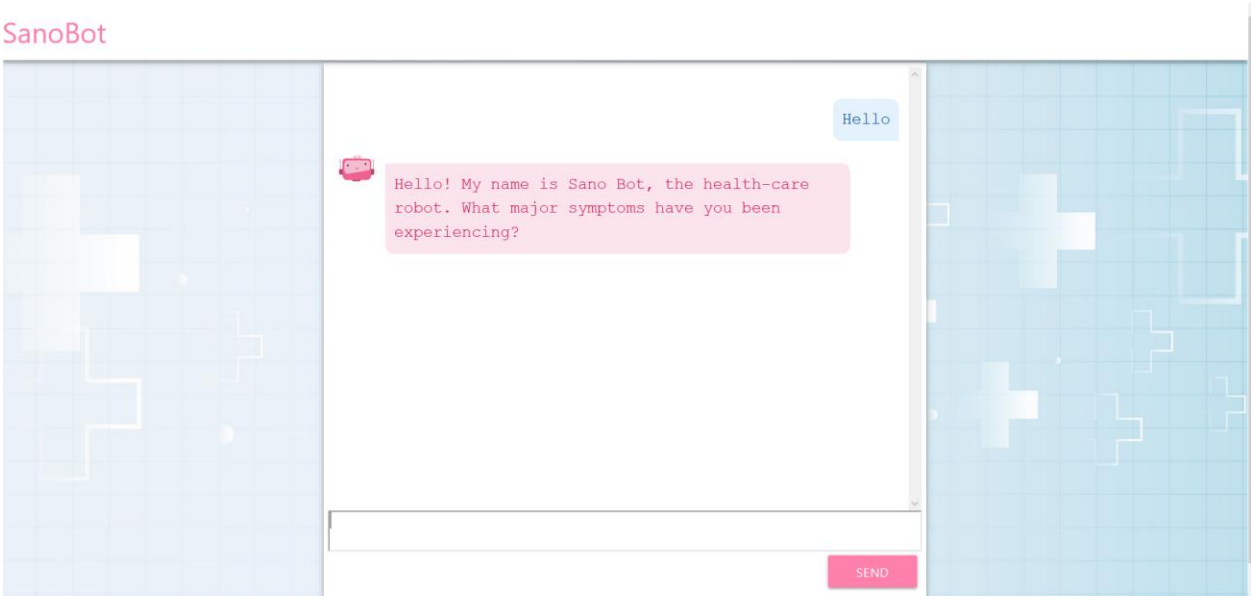


Figure 37: Chatbot SanoBot Interface

Dialogflow Backend

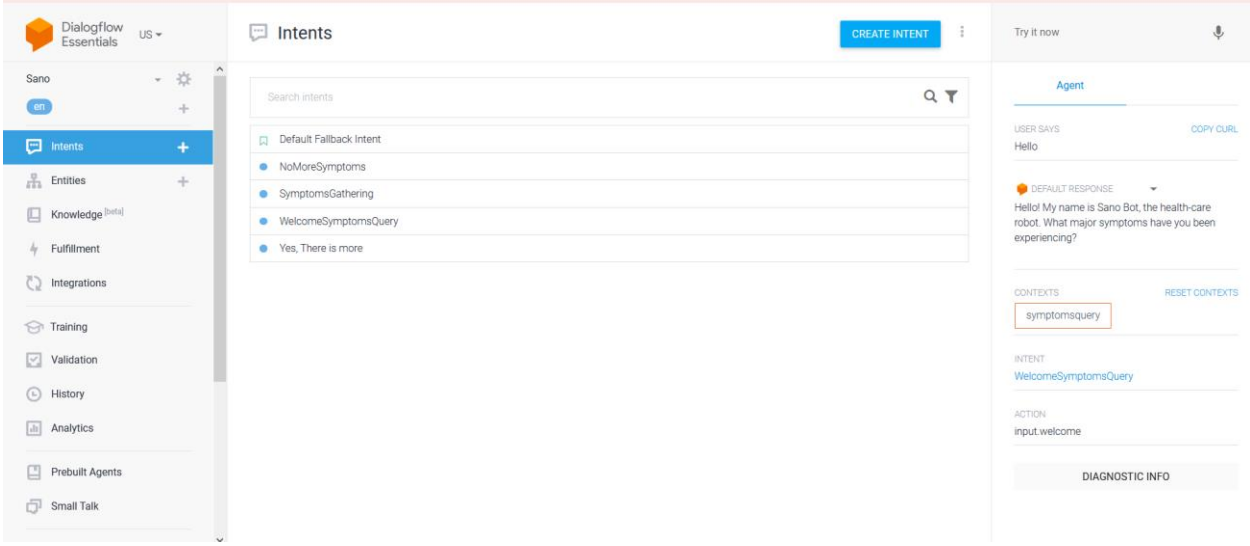


Figure 38: Dialogflow Backend

Connecting Dialogflow to Webhook

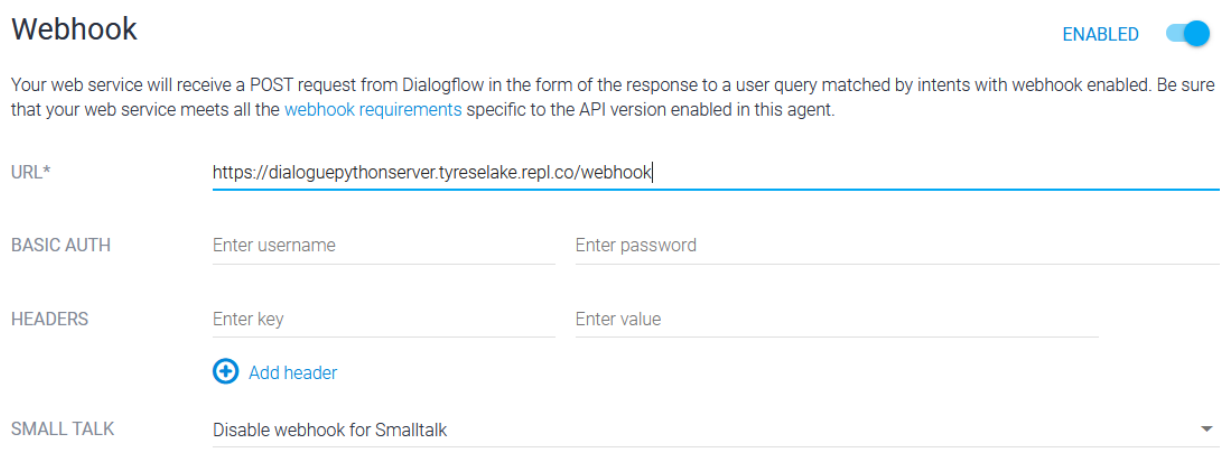


Figure 37: Establishing Webhook

References

Bahall, Mandreker. "Health Services in Trinidad: Throughput, Throughput Challenges, and the Impact of a Throughput Intervention on Overcrowding in a Public Health Institution." BMC Health Services Research. February 20, 2018. Accessed December 04, 2020. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5819239/.

Dialogflow. Dialogflow.cloud.google.com. (2020). Retrieved 4 December 2020, from https://dialogflow.cloud.google.com/.

Han, J., Kamber, M., & Pei, J. (2012). *Data mining*. Morgan Kaufmann.