BotMan

An enterprise-grade framework for chatbot development

Framework and Cloud deployment

Functional Specification

Version 0.1

Abhishek Madan
(abhishek.madan@sjsu.edu)

Version History

Version	Changes
0.1	First Draft

Introduction

A chatbot (short **for chat robot**) is a human developed computer program that engages a user in an interesting and result oriented conversation using tools for natural language processing to understand the users intent, and sends a response based on business rules and data of the organization. This data that the chat bot uses could either be host in a database or may be available over the internet in the form of Api responses. This data in its dormant state does not make much sense but when processes could give meaningful results. Understand human command and then extracting the useful information from the vendor database is what a chat bot is targeted towards achieving.

Botman is an enterprise-grade framework for chatbot development which enables a user to develop and maintain chatbot and dashbots for their personal and commercial use. The framework simplifies the development task by providing the user with a drag and drop interface and thus is easy to use for any user with no prior coding experience. It also takes care of deploying the bots on the cloud and managing its lifecycle. The framework also provides the user with the option to include one of the two NLP platforms namely Wit.ai and, Api.ai to improve bot response and engage users in a productive conversation.

While Chatbots seem to have a promising future, many things need to be taken care of to successfully configure and maintain it. Selecting a cloud infrastructure to host the chatbot is one of the crucial things to think about. The cloud platform should be reliable enough to maintain high service availability. Another consideration is selecting an NLP platform which could quickly understand the user's intention and reply appropriately. The model should be able to learn quickly and scale its knowledge starting with a limited dataset. More importantly, the chosen platform should handle error conditions gracefully.

This document discusses the function specs around developing such a bot framework. It systematically breaks down the entire framework in small modules and presents a stepwise functional analysis that is required to build the framework as a whole. The documents also talk about the test aspect to be considered while developing this framework.

References

1. https://www.smashingmagazine.com/2017/08/ai-chatbot-web-speech-api-node-js/

2. https://www.tutorialspoint.com/business_analysis/business_analysis_functional_requirements_document.htm

3. http://www.ofnisystems.com/services/validation/functional-requirements/

4. https://chatbotslife.com/ultimate-guide-to-leveraging-nlp-machine-learning-for-you-chatbot-531ff2dd870c

5. https://medium.com/google-cloud/using-google-cloud-emulators-for-integration-tests-7812890ebe0d

6. http://www.softwaretestinghelp.com/test-case-template-examples/

7. http://www.softwaretestingtimes.com/2010/04/step-by-step-guide-to-test-case.html

Requirements

Following is the high-level architectural diagram for the framework. It indicates how the various components interact to present the user with an environment that is required for developing and monitoring a Chatbot.

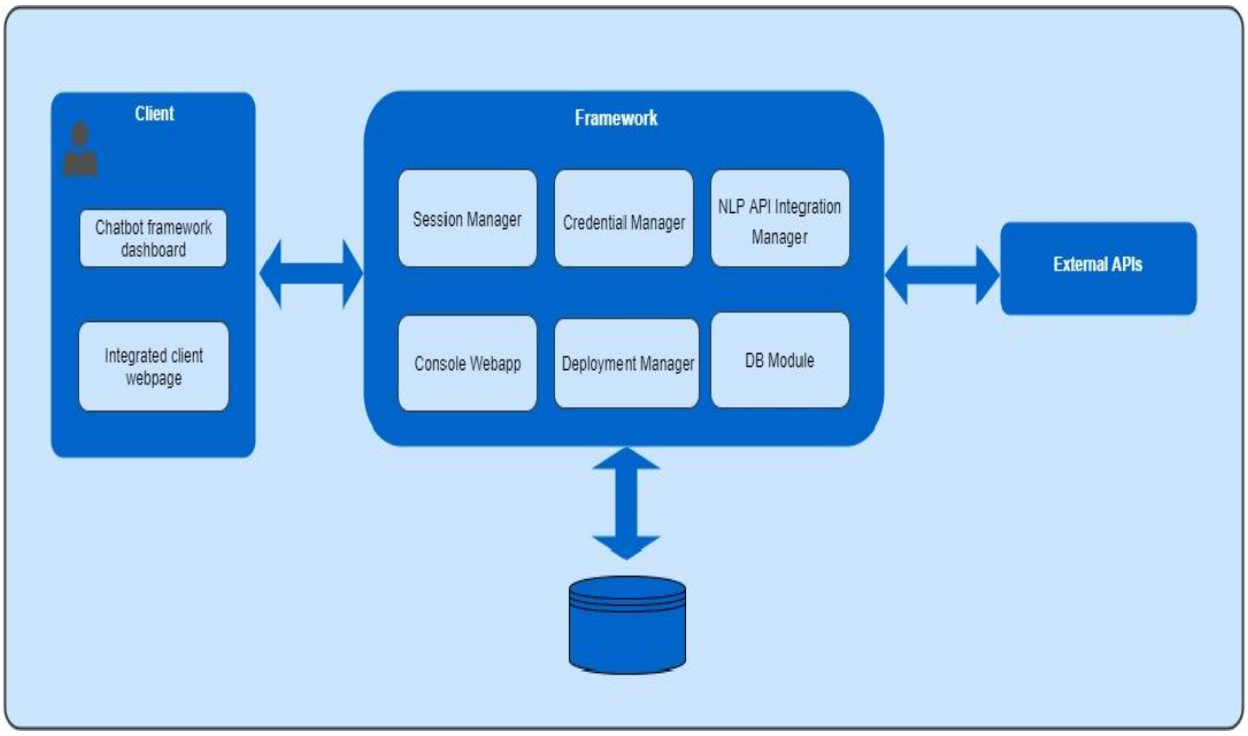

Fig 1: Architectural Diagram

In the above diagram, the framework basically includes the dashboard, hosted on a remote virtual instance, and a backend database to store the user info. This database stores information like the user details, the number of bots that the user has created till date, the NLP intent to questions mapping if the user has chosen to link any NLP platform with his bot or a set of question-answer mapping if the type of bot response is a fixed question to answer response, the cloud account credentials of the user where the bot developed by the user would be deployed.

The framework utilized the services provided by other SAS platforms to enable the users to add advance text processing and response capabilities. The framework provides the user the option to either add Wit.ai or api.ai as a natural language processing engines to identify users intent and respond appropriately. This NPL engines could be trained by using the NLP mapping interface present in the framework where the user maps the possible intents to some set of question and then relies on the NLP platform to predict the intent for any similar question asked by the user.

Some business operations may require querying the vendor's database to generate a response. The framework provides for an interface where the vendor can link his database service. Thus the user's query is first processed by a NLP to obtain the intent and this intent is then used to formulate a query

which gives the data from the database. The response obtained from querying the database is then posted to the user.

The vendor bot might in addition to querying his database might also rely on some third party Api. An example of this is when a vendor's bot might use the email service to send an email to the user it is talking to. To consider such case we have included an interface where a third party Api could be configured.

Lastly, we have the servers over which the bots would be deployed. We provide users with a list of options to choose from to deploy his chatbot. As the framework takes care of setting up of the environment and deploying the bot once the user selects the cloud platform, the user has a limited access to monitor the availability status of the bot. The framework thus provides a rich dashboard to monitor the status of the deployed bot. It also provides for a list of all the bots that the user wish to maintain and a control to deploy and pull down the bots.

Functional Process Requirements

Depending on the various functions that we intent to offer, the following functional requirements could be formulated-

Section ID	Requirement Description
FR1	The system shall allow user to maintain account information
FR 1.1	The system shall maintain user info and login credentials.
FR 1.2	The system shall maintain user's cloud account details.
FR2	The system shall allow user to create and maintain multiple bots.
FR2.1	The system shall allow the user to create multiple bots.
FR2.2	The system shall maintain all the different versions of the bots.
FR2.3	The system shall maintain the NLP mapping for intent and the phrase if NLP option is checked while creating the bot.
FR2.4	The system shall maintain the Question and Answer mapping if the Fixed response option is checked while creating the bot.
FR3	The system shall allow integrating different services available on the dashboard.
FR3.1	The system shall allow integrating one of the provided NLP services.
FR3.2	The system shall allow integrating a third party Api in the bot.
FR3.3	The system shall allow integrating the vendor's database service with the bot.
FR4	The system shall provide for controls to manage bots.
FR4.1	The system shall provide controls to start stop the bots.
FR4.2	The system shall provide the current availability status of the bots.
FR4.3	The system shall provide means to delete a bot.

Functional Overview

The framework would be implemented as a MEAN stack project which aims at integrating and enhancing the services provided by various SASS and PASS platforms. We plan to use the github/facebook login for account creation and management. The framework would be deployed on AWS and would be available as a service. The framework would be backed by a SQL/NoSQL database which maintains all the information for a user.

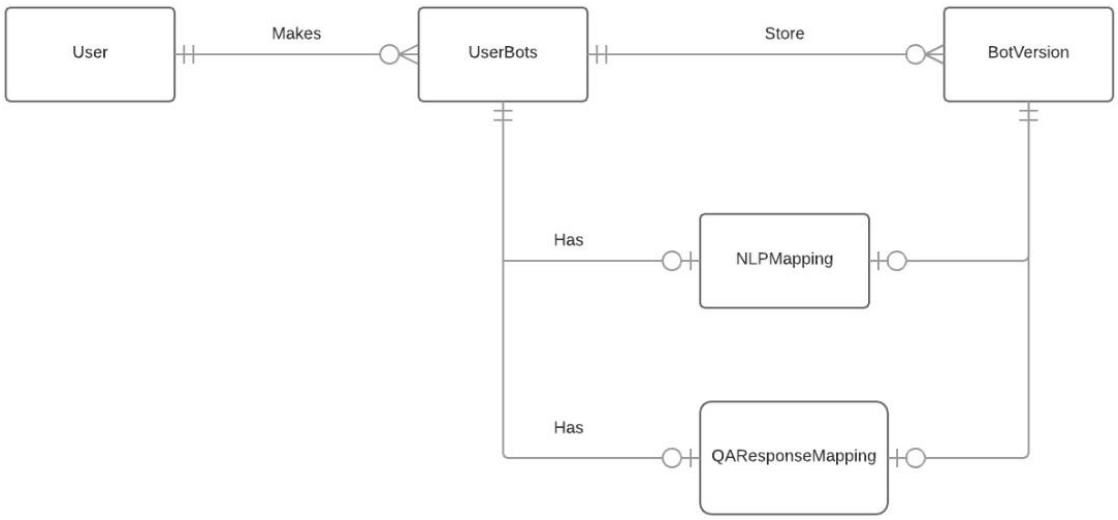

Fig 2: ER Diagram for DB schema

A user could make one or more chat/dash bots. These bots would be store in the database and at any time would have the latest bot configuration that is deployed on the cloud. Each bot can be configured either as a NLP backed bot or a simple fixed response to a question type of a bot. The mapping for these types of responses would be maintained in the database. Thus a bot could either be one of these and not both. The framework also maintains the version of the bot as it is updated. This would allow the user to revert back to some pervious response state.

The Chabot servers would be deployed on either IBM blumix or AWS depending on the cloud option a user chooses. This deployment would involve the respective platform apis for deployment and to monitor the availability status. The NPL mapping that the user configure would be used to train the NLP model on Api.ai or Wit.ai as per the user selection. All the fallback inputs, which the platform fails to classify, would be store in the local database and the user would be notified of any such event. This would enable the user to properly map these unclassified inputs with an intent and thus improve the natural language processing capability of the chat bot. The NLP service would also be accessed using the Apis that are made available by the respective platforms.

The framework would facilitate obtaining a dash bot which the user could then deploy on to his website. This Dashbot would include a set of files with code snippet for making requests to out

framework and a user interface that would appear as a widget on the client's website on deployment. The query to the interface would be send to the framework. It would be processed and the appropriate response would then be send back to the user. This would be a three-tier architecture where the client website would form the first tier, the bot server would be the second tier and, the associated third party services would form the third and the final tier.

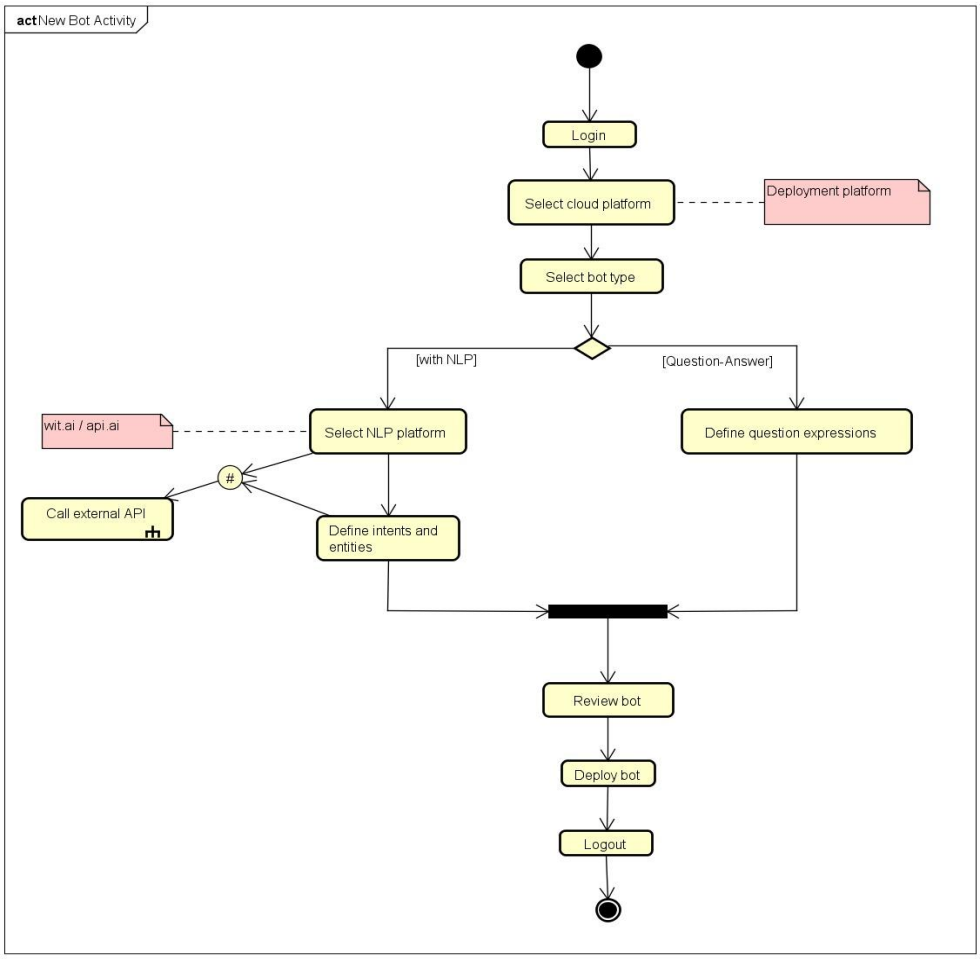

Fig 3: Flow diagram

The above diagram is the function flow diagram representing the various states the user go through in the framework depending on the type of bot they wish to create.

Configuration/External Interfaces

The framework relies on a number of external services and interfaces to accomplish various functionality that is required for creating and deploying a chat bot. These external services are as listed below-

- Web deployment services like AWS, IBM blumix for deploying the Chatbots.
- NLP services like Api.ai and Wit.ai for text processing.
- Third party Api's that the vendor would explicitly like to link in the chat bot response.
- Configuration around the dash bot deployment on the client side.

Debug

The debugging for error correction needs to be employed at the framework level as well as at the various services that are being integrated into the framework. Cloud deployment forms an integral part of the project and so amazon's cloud watch tool would be used to monitor the traffic over the network. At the framework level we would use the node's *'Debug-logger'* module. It is an ubitiquous logging library with 1000+ dependents. Given how widespread it is and the convenience of namespaces it is a great logger for library modules. Debug-logger is a convenience wrapper around debug that adds level based coloured output. Each instance of debug-logger will lazily instantiate several instances of debug such as namespace:info, namespace:warn, namespace:error, etc. All these are configurable. Debug-logger has no dependencies besides debugging, which is built right into node package.

Logging

Supporting multiple user login involves keeping a track of user activities so as the efficiently and quickly identify the point of failure if any. Since the framework would be made available as a service over the internet, network logging services like Data dog, Foglight or Logz.io. These are cloud infrastructure monitoring service, with a dashboard, alerting, and visualizations of metrics which makes monitoring network traffic easy.

Implementation

The framework implementation would involve working on the following front.

User data and session management

This includes saving the user credentials while creating an account on the portal. As the user uses the framework for developing bots, the platform must provide them the facility to save multiple bots. These bots should be available at all times when the user wants to review their behavior. Some bots would be QA based bots. These bots would give a fixed response when asked a fixed question. The question and answer mapping for these bots is required to be stored in the database. Yet another type of bot is an NLP based bot. This bot would rely on the input to response mapping and this mapping would also be stored in the database. The user can review and modify these configuration/mapping when he desires to change the response of his bot. The framework must also store the different versions of the bot as the user makes changes. This would help the user to revert to any previous functional state he wishes to.

Dashboard development

Every user who creates an account gets a dashboard view which he could use to manage their bots. This would be the default view and would indicate information like the number of bots the user has created, the number of bots active at the time the user sees the dashboard, a panel to monitor the health of the bot, and setting to deploy or un-deploy the bot. Every time a user creates a new bot, that bot would appear on the dash bot as a details row giving different information about the bot. deleting a row would delete the chat bot and would also pull it down from the deployed server.

Training a NLP model

The framework would allow a user to integrate NLP capabilities while creating a bot. These services would be made available as services from different platforms. At the beginning, the framework would make available the NLP services provided by Wit.ai and Api.ai for integration. In order to train these models the user will provide a list of mappings between the inputs that the chat bot could is expected to receive and the intent to which these inputs needs to be map to. There may be one or more inputs map to a single intent and many such mappings could apply to a bot. These mappings are then sent to the respective NLP platform service that the user has selected to configure in the chat bot. The mapping of the intent and the inputs could change as the user realizes that the current mapping is not adequate enough to generate a proper response. Any update in the mapping would then be resent to the NPL platforms to retrain the existing models. An example of NPL mapping is as show below-

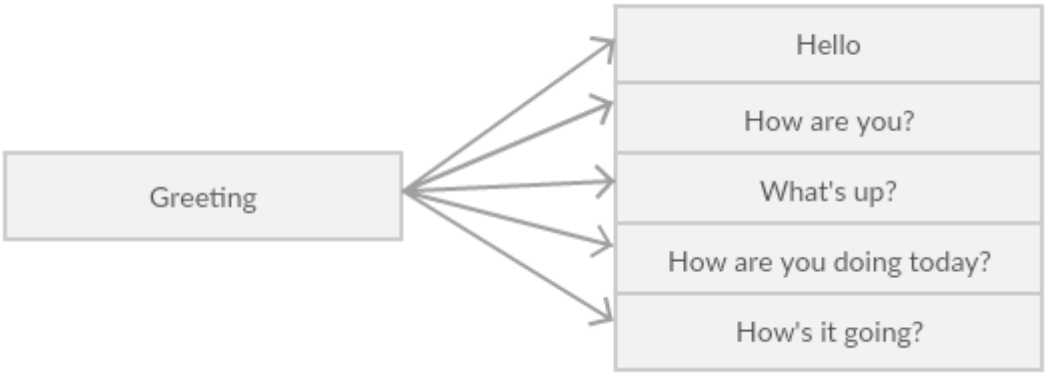

Fig 4: NLP mapping example

Third party and vendor database Integration

One of the prime requirement of any vendor is that the when the customer asks a question, the person responding to it should take in account all the previous transaction that are done on that issue/subject. These transactions are stored in the vendor database. The database should be easily integrable with the bot. This involves identifying the columns in the database that need to be queried based on the intent that is obtained from the NLP platform. Thus the complete flow could represented as follows-

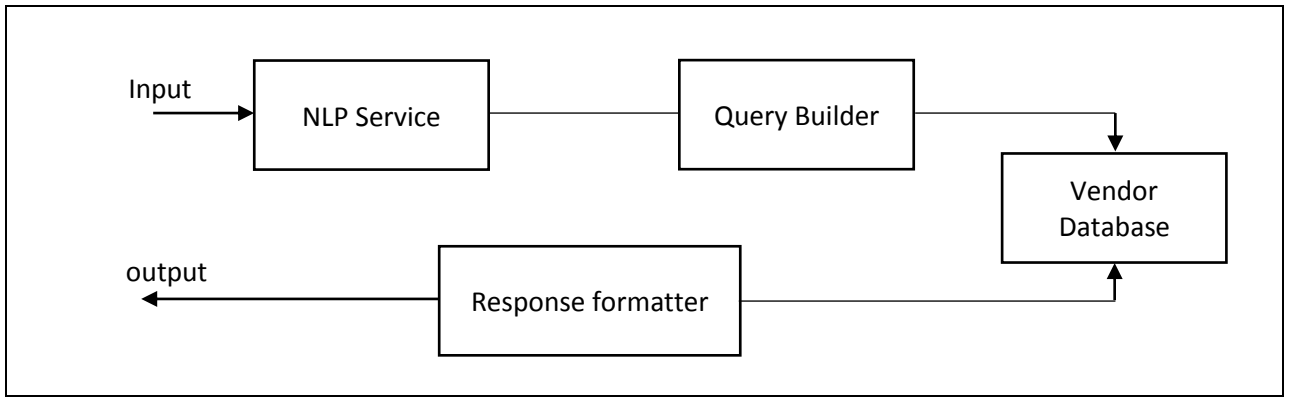

Fig 5: Vendor database integration with bot

Testing

General Approach

Software testing is one of the most important phases in the process of software development. Its primary objective is to ensure quality. This includes validation and verification both. Validation is checking if the developed software complies with all of the standards and checking if the entire user requirements are met. The main focus is to uncover as many bugs as possible. Being a systematic process, software testing requires the use of standard test rules and methods.

The scope of testing not just covers testing the application developed but also documenting and reporting it. Documenting its helps future developers and testers understand the functioning of the application, what parts of it have been tested, if tested thoroughly or not. Every error or bug that has been discovered in the application must be reported and documented. Next step in the process is to fix the reported bug. Once these changes are made, the complete process must be documented. To conduct all these, test cases are employed record details like test case number, name of the tester, type of test, preconditions for the test , operations to be performed ,input data and expected outcome. Unit Tests List of unit tests that were done as a part of this functionality.

In our project we would be using the following periodic three step testing approach-

- Unit tests: It includes the tests on basic functionalities that the framework offers. Each unit test would cover one and only one. We would create proxy stubs for mimicking models and use chai and mocha for test case validation.

- Integrations test: It includes combining the functionality offered by individual module and verifies that the end to end flow is met.

- Performance test: This includes testing the framework performance when multiple user requests are made at a same particular time. We plan to user Jmeter for load performance testing. By evaluating the max load performance on a single machine we can decide on the scaling factor to improve the availability of the service.

Unit Tests

For each of the sections discussed in the implementation section, the following are the unit test cases-

User data and session management

S no.	Test case Id.	Objective	Expected Result
1	T1.1	User account creation	The user should be able to create a new account.
2	T1.2	User login	The user should be able to login in to his account using his account credentials

3	T1.3	User session management	A separate session should be created for each user which keeps track of user activity.
4	T1.4	User bot history management	The framework should successfully display all the chat bots created by the user in the past.
5	T1.5	Allow updating a bot	The user should be able to update the bot response by change the input to response mapping and should be able to save this to the database.

Dashboard development

S no.	Test case Id.	Objective	Expected Result
1	T2.1	Show all bots created by the user.	The dashboard should show all the bots created by the user till date and this should be the landing page for a user after login.
2	T2.2	Bot controls on dashboard	The dashboard should provide controls for managing the lifecycle of the bot and also provide controls to add or delete a bot.
3	T2.3	Bot availability status control	The dashboard at any time provide the real-time status of the bot.

Training a NLP model

S no.	Test case Id.	Objective	Expected Result
1	T3.1	Display mapping for NLP bots	The framework should display the intent to input mapping for all the bots that has NLP configuration.
2	T3.2	Alter NLP mapping	The framework should provide for a convenient way to the user to modify the existing intent to input mapping.
3	T3.3	Update the Model with the new NLP mapping	The framework should automatically send the new mapping to the NLP services for training the model.
4	T3.4	Having a fallback intent.	The framework should gracefully handle all the inputs for which no intent could be found. These inputs are later presented to the user to categorize then in one of the mapping or create a new intent for these types of inputs.

Third party and vendor database Integration

S no.	Test case Id.	Objective	Expected Result
1	T4.1	Easy integration of vendor database.	The framework should provide for a convenient way to integrate vendor database. The chat bot can query this database to formulate response based on the previous business transactions.
2	T4.2	Easy integration of third party Apis.	The framework should provide for a convenient way to integrate any third party Api. These Apis could be used for a fixed question type real-time response.