****

**Sayso –** Survey Bot of **ITC InfoTech India Limited**

**Overview:** A chatbot application to conduct surveys by asking questions to users and store user response in the persistence storage for future analysis.

**Bot Project Structure:** Following is the structure of the **Sayso Bot** application;

**I3L.ISURVEY.BOT.WEB**

**Authentication**

**Config**

**Dialogs**

**DoSurvey**

**Error**

**Feedback**

**FormatSurveyQ&A**

**RemindMeLater**

**RespondCard**

**StartUp**

**SurveyList**

**WebAPI**

**server.js, env, gitignore, package.json, Readme.md, etc.**

**Module Descriptions:**

* **Module Name** **:** Authentication
* **Purpose :** Validate and identify user from Bot Service
* **Steps** **:**
  1. Get **GUID** from bot/iframe URL by accessing **userid** parameter. From Bot side it is accessible via **session.message.user.id**.
  2. Call Azure function to get Encrypted SharePoint Token(**ESP Token**) against the GUID (**AES-CBC-256** Algorithm is used here).
  3. Decrypt the ESP token and get Original SharePoint Token(**SP Token**).
  4. Get logged in user details by calling **Café API**(production) using **Authorization** header as SP Token (**Authorization = {SP Token}**)**.**
* **File(s) :**
  1. **identifyUserViaSharePointToken.js** -- call Azure Function to get ESP Token, call Café API to get user details, decrypt the ESP Token and store user informations (name, psid, email, manager, etc) in Bot’s memory.
* **Module Name :** Config
* **Purpose :** Hold the API configurations(**URIs**) of **development**, **staging** and **production** environment.
* **File(s) :**
  1. **default.json** – Hold default API configurations
  2. **development.json** – Hold **dev API server’s (IIS)** configuration.
  3. **staging.json** – Hold staging API server’s (**MSWEB**) configuration.
  4. **production.json** – Hold production API server’s configuration.
  5. **custom-environment-variables.json** – Used to hold secret informations like passwords, keys. In this bot, nothing is used as such.
* **Module Name :** Dialogs
* **Purpose:** This module contains all the dialog functions of each dialog. Each file of this module have an array of functions which contains the attribute and behavior of that dialog. Based on the requirement, this module redirect the control flow to other modules like **RemindMeLater**, **doSurvey**, etc.
* **File(s) :**
  1. **root.js** – Hold Array of functions of **root** dialog.
  2. **startSurvey.js** -- Hold Array of functions of **startSurvey** dialog.
  3. **gridChoiceTypeQuestions.js** -- Hold Array of functions of **gridChoiceTypeQuestions** dialog.
  4. **beginAllSurvey.js** -- Hold Array of functions of **beginAllSurvey** dialog.
  5. **botFeedback.js** -- Hold Array of functions of **botFeedback** dialog.
* **Module Name :** DoSurvey
* **Purpose :** This module is responsible to ask questions to user, store user responses, keep track of current question and corresponding branching logic, submit survey response periodically, check for survey completion status and once completed submit the final response.
* **File(s) :**
  1. **gridChoiceTypeQuestion.js** – Handle survey questions of type “**gridChoice**”, keep track of all sub-questions, store answer of each sub-question, and finally merge answers of all sub-queries and store them in response array as a single answer.
  2. **partialSurveyQs.js** – Look for the active surveys whose **status** are “**saved**”, get hold of the partial submitted response, look for the answer of last submitted question, analyze its branching logic of that question, find the next question to be asked and then resume the survey.
  3. **promptUserBasedOnQuestionType.js –** Every survey question has some **response type**, like **boolean**, **choice**, **multichoice**, **gridchoice**, **text**, etc. based on which the question is presented to user. These **type checking** is handled in this file.
  4. **surveyQ&A.js** – This is the primary file of this module. This file handles following operations;
     + Get Survey questions by Id.
     + Start the survey by asking questions.
     + Store each response of user.
     + Check for the survey completion status.
  5. **submitSurveyResponse –** This file is responsible to submit response to sharepoint via web API. Following are the responsibilities of this file;
     + Submit response periodically with survey **status = saved.**
     + Once survey gets completed, submit the entire response with survey **status=completed.**
  6. **takeAllSurvey.js –** This file is responsible to conduct **all active surveys** one by one. When user selects to take all survey, then this file initiate the surveys one by one, keep track of completion status of each survey and when all surveys are done then end the conversation with appropriate message.
* **Module Name :** Error
* **Purpose:** This module is responsible to **log errors** which happens (if any) during runtime of the bot application. Bot logs every error to **Azure Table** via calling an **Azure Function**. Bot keep track of every application errors via predefined **error events** namely “**uncaughtException**” and “**unhandledRejection**”, whose handler are defined in **server.js** file. From these handler function, this error module is getting invoked.
* **File(s) :**
  1. **logError.js –** This is one and only file of this module. This file have only one method named “**logErrorToServer( err, stackTrace)**”, which logs error to Azure Table with **error message** and complete **stack trace** of the error.
* **Module Name :** Feedback
* **Purpose:** The main intention of this module is to get user **feedback** about this **survey procedure,** in simple term this module is responsible to take Bot’s feedback.
* **File(s) :**
  1. **feedback.js –** Prompt user to give feedback by displaying a feedback card, accept user feedback, accept additional comments (if any) and then bid goodbye and end the conversation.
* **Module Name :** FormatSurveyQ&A
* **Purpose:** “**promptUserBasedOnQuestionType.js”** fileof module “**doSurvey**” is responsible to display questions in front of users based on the **response type** of question. In order to do that, first bot needs to put every question and corresponding answer options in a card view which is known **adaptive card** in bot framework. This module is mainly responsible to put the **question and answer options** in appropriate adaptive card and then return the card to “**promptUserBasedOnQuestionType.js”** file for rendering.
* **File(s) :**
  1. **choiceInputOption.js** – Put question and corresponding answers as suggested action and then return the suggested action message.
  2. **dateTimeInputOption.js** – Put question in a date field enabled adaptive card.
  3. **dropDownListInputOption.js** – Put question in card and list down the possible answer options as dropdown.
  4. **multiValueInputOption.js** -- Put question in card and list down the possible answer options as checkbox.
  5. **numberInputOption.js** – Put question in card and prompt user to enter a number mainly integer.
  6. **textboxInputOption.js** – Put question in card and add an input **text field** to card to allow users to enter answer as text.
* **Module Name :** RemindMeLater
* **Purpose:** When user wants to take the surveys at a later point of time, then this module is getting invoked. Primary goal of this module is to inform the backend ( **Sql Server DB**), about the next reminder date of each active survey. When next reminder day will come, then only bot will show those surveys again to user.
* **File(s) :**
  1. **remindMeLaterResponse.js** – This is the only file of this module. This file is responsible to perform following tasks;
     + Calculate the next reminder date by looking at “**RemindMedays”** attribute of active **survey objects**.
     + Store all survey along with their next reminder date as an array in Bot’s memory.
     + Post the above array in **RemindMeLater** web API.
* **Module Name :** ResponseCard
* **Purpose:** This module holds all the necessary **adaptive cards** for different types of questions. Based on the survey question’s **response type**, appropriate adaptive card gets returned from this module.
* **File(s) :**
  1. **checkboxCard.js**
  2. **dateTimeCard.js**
  3. **dropDownCard.js**
  4. **intoCard.js**
  5. **numberCard.js**
  6. **textboxCard.js**
* **Module Name :** StartUp
* **Purpose:** This module mainly contains all the **server configurations** and **initialization** related functionalities. Also this module holds few basic **middleware** functions.
* **File(s) :**
  1. **auth.js** – It is a middleware function object, to initiate **user authentication** at the very **beginning of the conversation.** When **node server** starts, this is the first function which gets executed.
  2. **initiateConversationFromBotSide.js** – A function which executes when bot gets initialized and the purpose of this function is to initiate the conversation from Bot’s side.
  3. **serverConfiguration.js** – This is a file based module which holds all necessary configurations like listening to specific port, post message URI, front-end setup, etc. related to **node server**.
  4. **sessionTimeout.js** – This file is responsible for **session timeout management**. It is also a middleware function which keeps track of user’s **last interaction** timing, and ends the current session if user’s idle time exceeds the configured timeout duration.
  5. **typingIndicator.js** – Simple middleware function to show a typing indicator to user whenever there is some delay in displaying messages.
  6. **Index.html** – This is the **host file** of this application. This is the file which appeared first whenever the application is accessed on the web.
* **Module Name :** SurveyList
* **Purpose:** This module is responsible to **display the list of active surveys** to the user along with “**Take All**” and “**Remind Me Later**” option. Whenever user access this bot, after successful authentication this module gets invoked. Also based on user’s initial selection, this module invokes other modules like, if user selects “**remind me later”** option, then the “**RemindMeLater**” module gets invoked.
* **File(s) :**
  1. **activeSurveyList.js –** This file module is responsible to perform following tasks;
     + Get Active Surveys by calling API.
     + Store all surveys along with their “**RemindMeDays**” duration.
     + Display active surveys to user along with “**Take All**” and “**Remind Me Later**” option.
  2. **displaySurveyAndDoOperations.js** – Accept users initial choice( **specific survey/ take all/ remind me later**) and based on that invoke appropriate module.
* **Module Name :** WebAPI
* **Purpose:** This module contains all the **web api** request parameters which are used in this application.
* **File(s) :**
  1. **activeSurveySummaryAPI.js –** Holds request parameters to **get** active survey list.
  2. **authAPI.js –** Holds request parameters for user authentication.
  3. **remindMeResponseAPI.js –** Holds request parameters to **post** remind me data.
  4. **submitSurveyResponseAPI.js –** Holds request parameters to **post** survey response.
  5. **surveyDataAPI.js –** Contains request parameters to **get** survey questions by id.

**Other Important Files** :

* **server.js** – This is the starting file of the application. This file is responsible to create a server instance, create a bot connector, create and initialize bot instance, initialize middleware functions and initiate all dialog flows.
* **.env** – Environmental file which holds different **environmental variables** like test tokens, partial response time interval, session timeout interval, etc.
* **.gitignore** – A file contains the file/directory name which should not be considered when storing this application in any remote repository like github, bitbucket, etc. This file contains “**node\_modules**” directory name, which should be removed while creating a remote repository.
* **package.json** –Manifest file of the application which contains some generic information related to the application like name, version, authors, etc and keeps track of all the dependency modules on which our project is **directly** dependent.
* **package-lock.json** – Manifest file which contains different informations about all the third party **node modules** on which our application is **directly/indirectly** dependent.