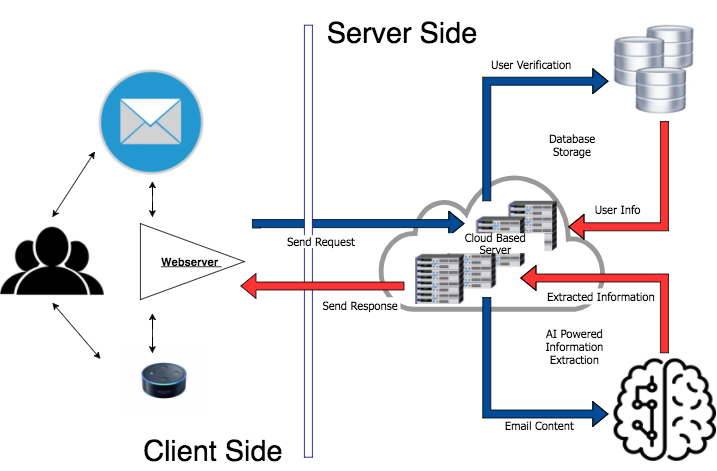
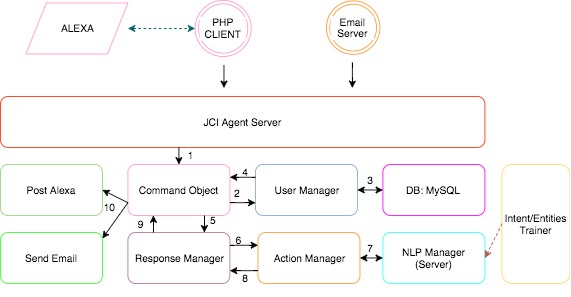
Nikhil Chakravarthy

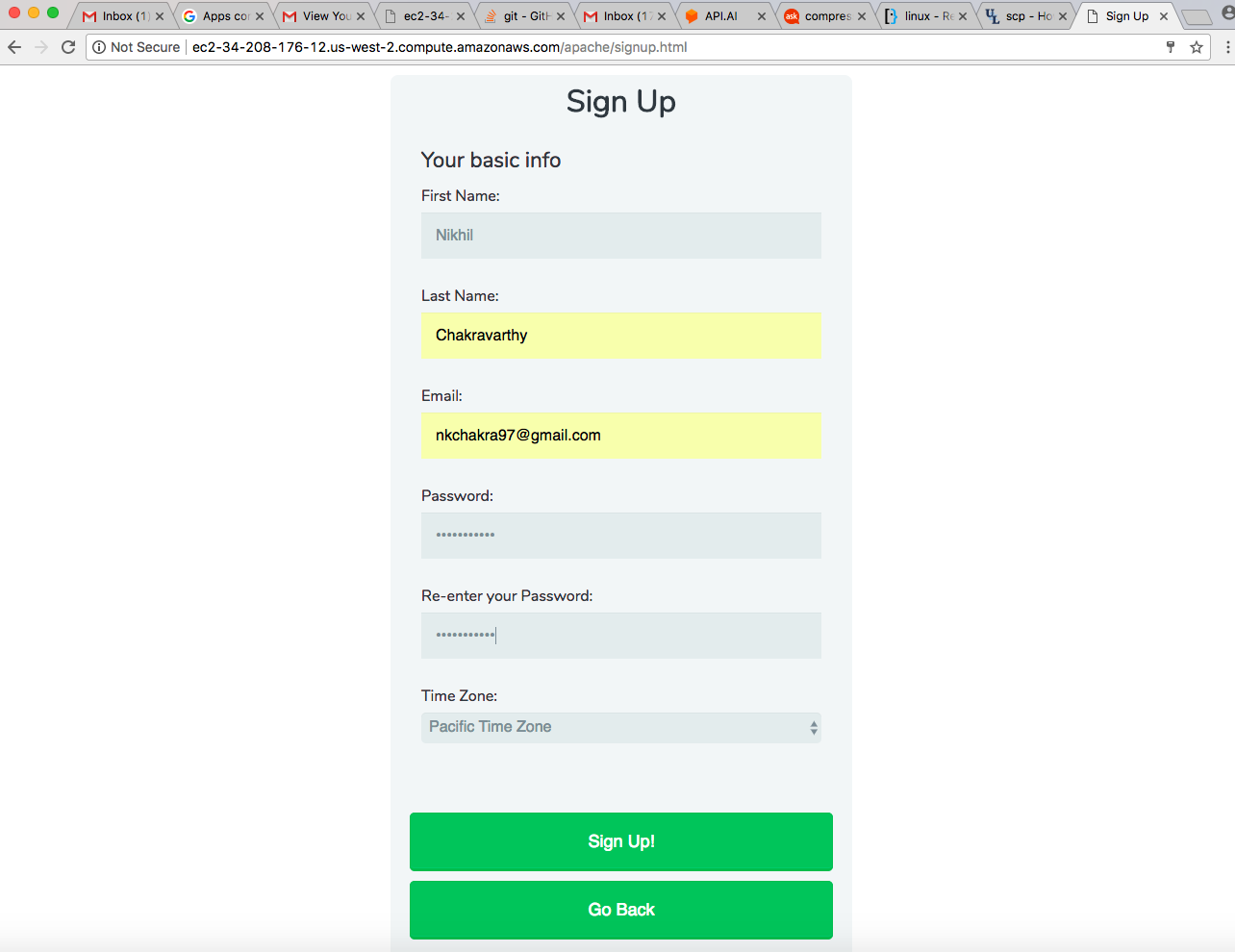
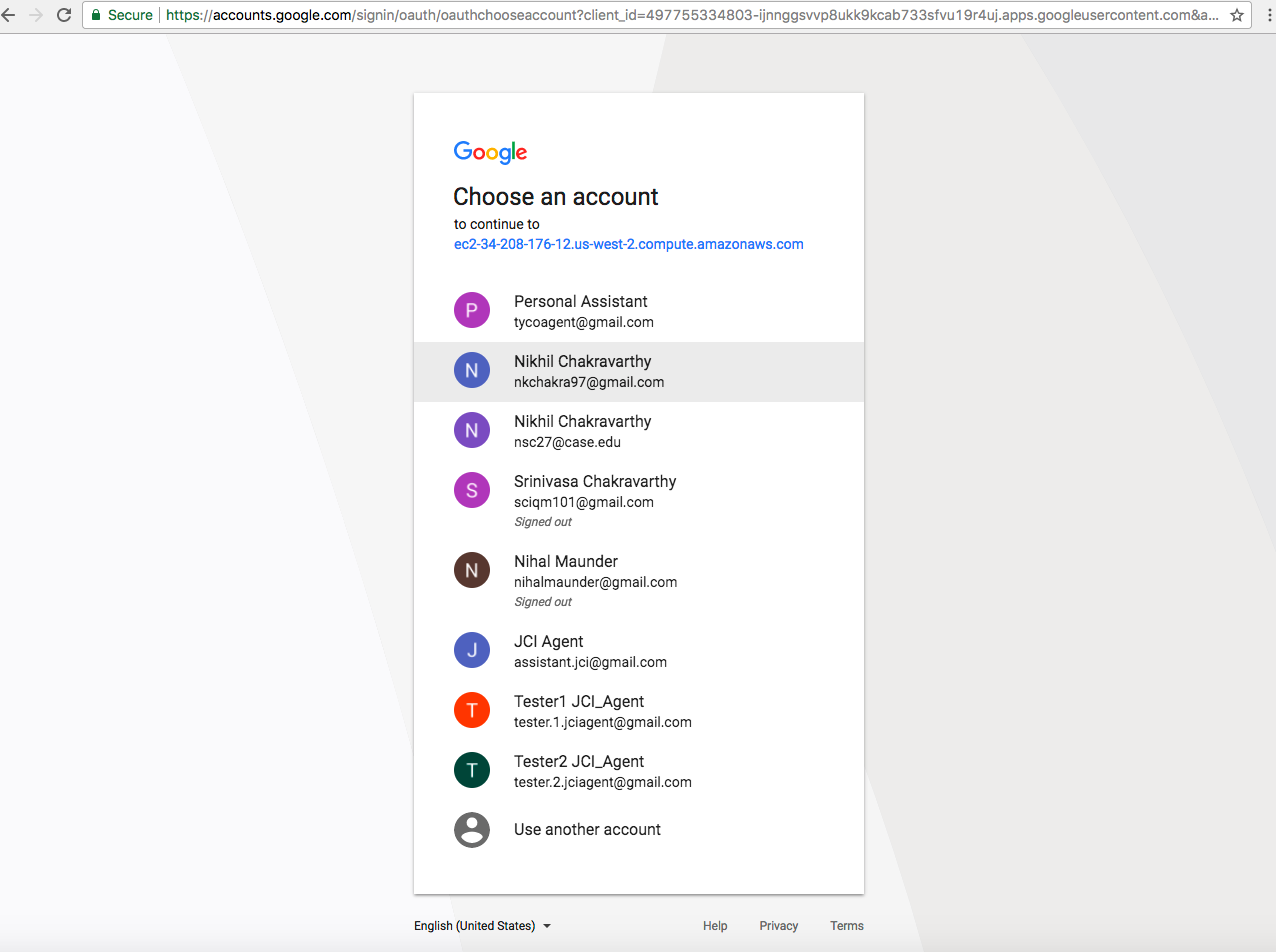
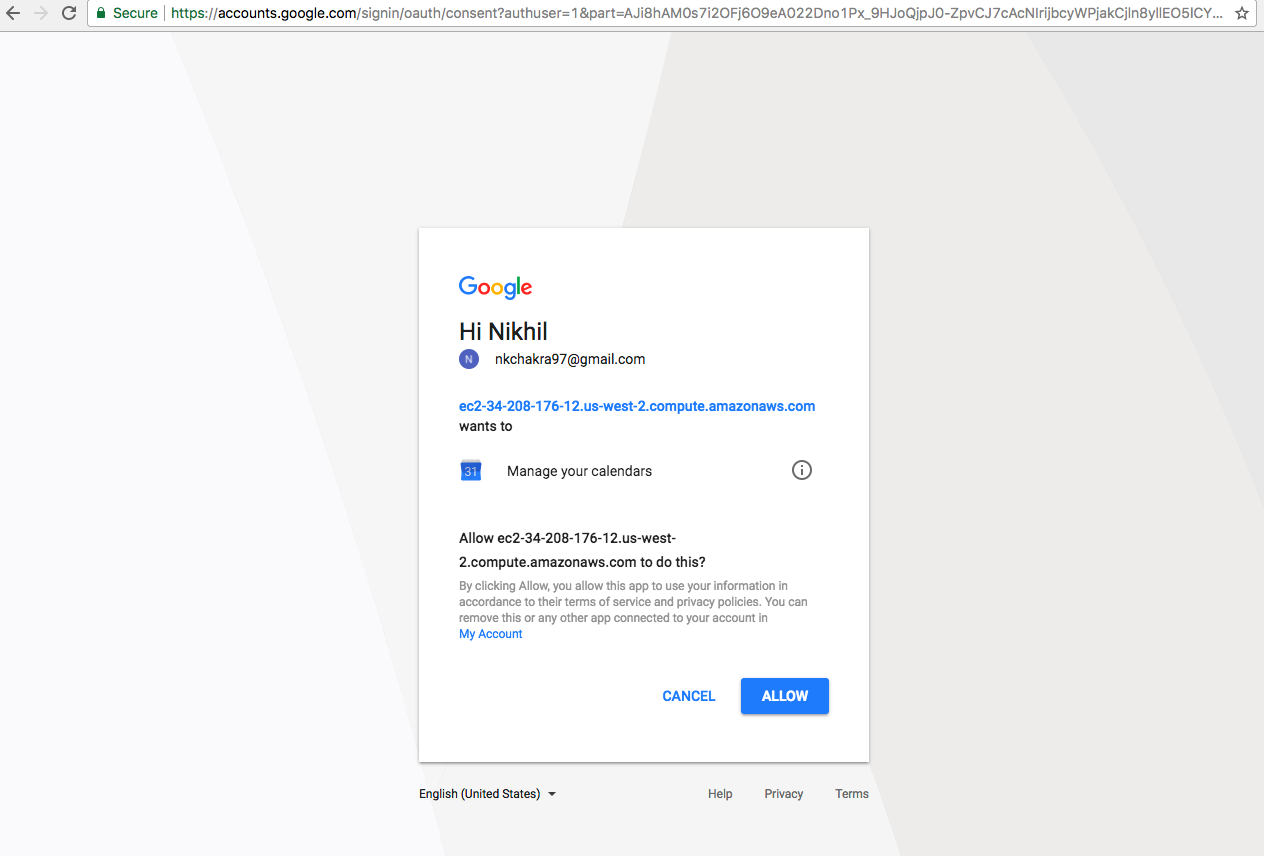
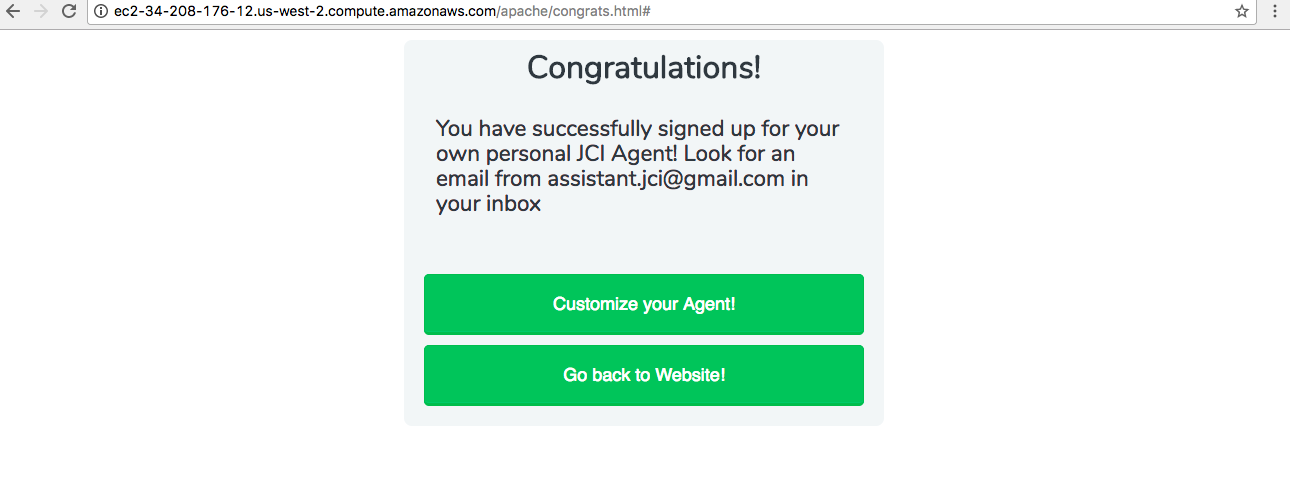
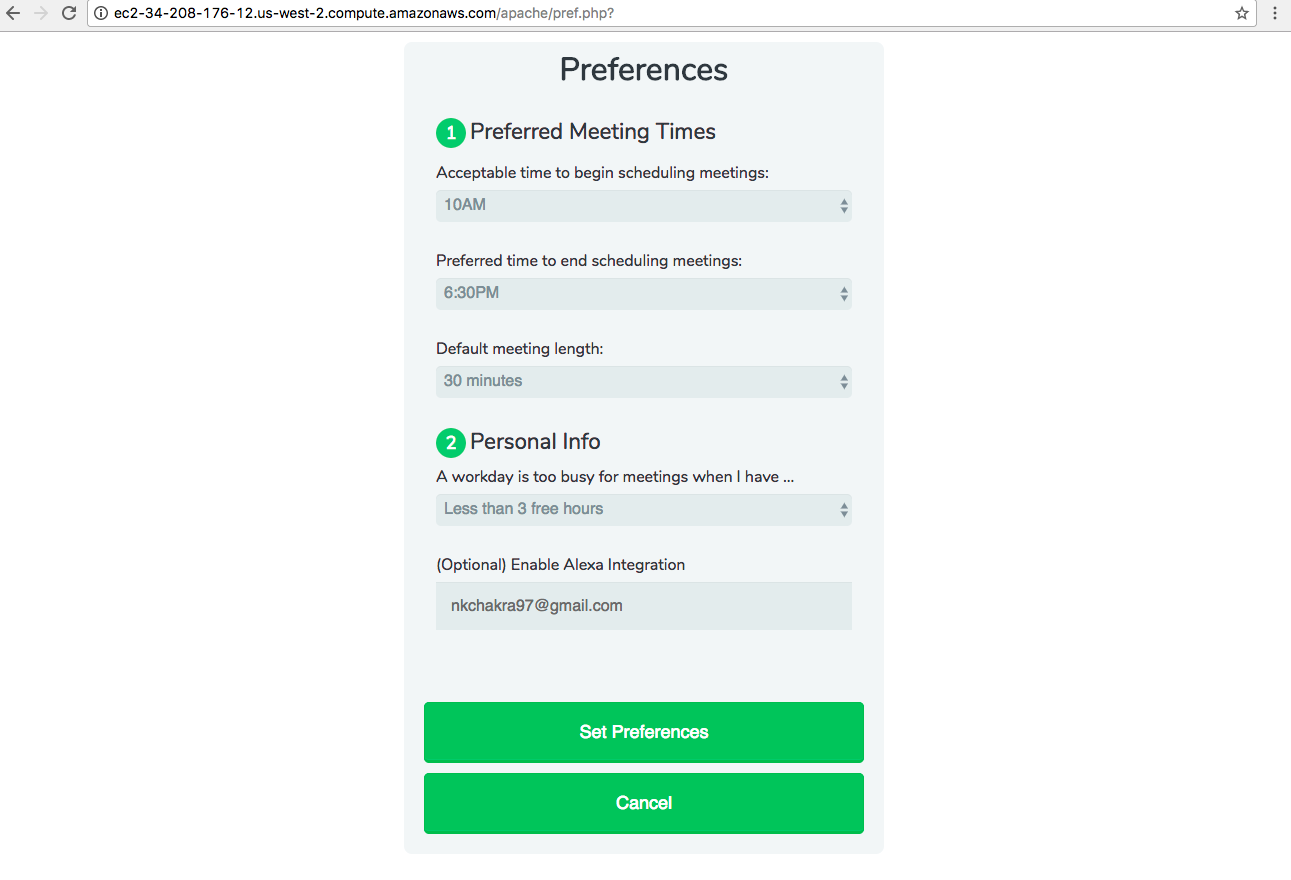
README Documentation for JCI Agent Project

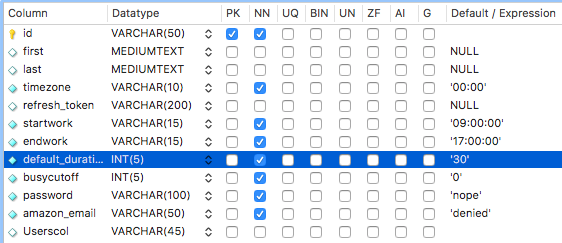
1. Overview
   1. High Level Architecture
   2. Low Level Architecture of each of the 3 concurrently running programs
   3. Website Architecture
   4. MySQL Setup
2. NLP
   1. Using API.AI
   2. How to switch NLP service/program
3. Email
   1. How it currently works
   2. How to add more interfaces
4. Calendar
   1. How it currently works
   2. How to add capabilities
   3. How to add more interfaces
5. Servers/Clients setup
   1. Ports
   2. JSON formatting and JSON-friendly objects
   3. Server Threading
6. Alexa
   1. Requirements
   2. Setup
   3. How to demo
7. Demo/Project Setup
   1. Testing Gmail/Amazon accounts
   2. Authorize via website
   3. Porting from Github
8. FAQ
   1. Capabilities currently
   2. Next steps
   3. Current Scalability

**OVERVIEW**

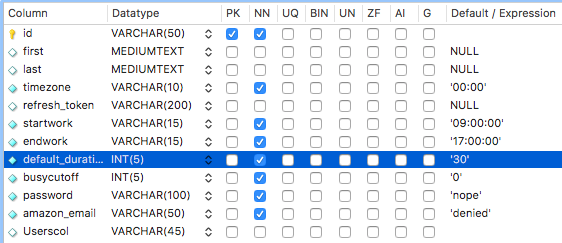
1. Introduction
   1. JCI Agent is a Cognitve Agent that can understand everyday talk by using NLP, and can carry out actions on a user’s calendar
   2. The current NLP service we use to figure out what our users are saying is API.AI
   3. JCI Agent works on email and on Alexa
2. High Level Architecture
   1. There are 3 concurrently running programs- tyco\_server, api\_requester, email\_server
      1. Tyco\_server is the backend server that takes in json command objects and carries out the request stored in that json object
      2. Email\_server is the server that uses the Gmail RESTful API to get all the new valid emails from the [assistant.jci@gmail.com](mailto:assistant.jci@gmail.com) account every 5 to 30 seconds (can be set in the source code)
      3. Api\_requester is a server that waits for client calls from the tyco\_server backend to send a query to API.AI. We do this to keep the connection to API.AI live, making it way quicker than if we didn’t
   2. The Alexa lambda function is hosted at a different AWS instance, and writes to the tyco\_server port directly
3. Low Level Architecture
   1. Tyco\_server
      1. First, it receives a client connection to the server, then spawns a thread to handle that client request
      2. The thread reads from the socket, reading a JSON version of a command object, and recreates the Command object
      3. Then a request to the api\_requester server is made
   2. Api\_requester
      1. The api\_requester server functions similarly to the tyco\_server. It waits for a client connection from the tyco\_server backend, then spawns a thread to handle the request
      2. The thread reads a json version of the NLP\_Response object(different from Command object), reads it into an actual object, then sends the request to the API.AI server to analyze it
      3. Then once the API.AI server response with a result, the important parts are put into the NLP\_Response object, which is then turned into a json to be written back to the tyco\_server
   3. (Back to tyco\_server)
      1. Once the NLP\_Response object has the information from the api\_requester server, the backend generates a text response and does the required action
      2. Then, according to the type of Command the server took in, the thread will terminate in either writing a json back to the client socket if it was an Alexa-based command (the client is the AWS lambda function), or if it was an email-based command, the socket is closed and the email is generated and sent to the appropriate parties
   4. Email\_server
      1. The email server checks the [assistant.jci@gmail.com](mailto:assistant.jci@gmail.com) email account for new emails every few seconds (can be set in the source code)
      2. It takes the new emails, sees which ones are spam or not normal emails and removes them
      3. Then it checks each of the new emails to see if the email is either from a registered user or part of a correspondence for a certain registered user (using thread id). If it meets the criteria, it will be send to the tyco\_server as a json version of the Command object. If not, an unauthorization email is sent back to the sender with a link to the website homepage



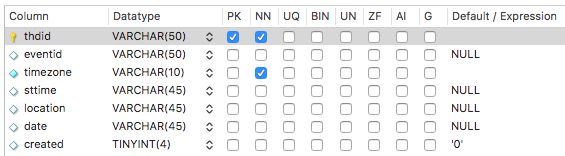
1. Website (http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com/apache/)
   1. \*\*NOTE\*\* The website is currently very barebones and relatively unorganized. All the .html and .php files are under the folder apache (/var/www/html/apache/), not organized into directories
   2. The important php and html files are located in the website folder of the project
   3. The website has some basic info on it, as well as a crude Sign-Up and Log-In function.
   4. Only the Sign Up and Log In buttons actually lead to another page from the home page
   5. When someone Signs Up, php code is run to to redirect the person to Google’s Oauth 2.0 authentication system. \*\*NOTE\*\* the ‘vendor’ directory needs to be under the same directory as the ‘initialauth.php’ and ‘secondaryauth.php’ files, since they use it to get Google API methods
   6. When they authorize JCI Agent on Google’s side, a refresh token is received by the php files, and inserted into the database, along with the person’s name, email, and timezone. The password is passed as an argument and called in a java jar, which salts the password and one-way hashes it
   7. Then the user has an option to set preferences. If they do, php code is run to put the preferences in the database
2. MySQL Database
   1. The MySQL database is hosted locally on the same AWS as all the servers, and is called ‘Tyco\_Agent\_Schema’
   2. It has the username ‘root’, with password ‘CognitiveAgentDB’ at port ‘3306’
   3. It has 3 tables inside it
      1. Users – primary key: id (email address)
         1. This table stores all the user info such as name, email, hashed password, refresh token (for their Calendar), preferences, and timezone.



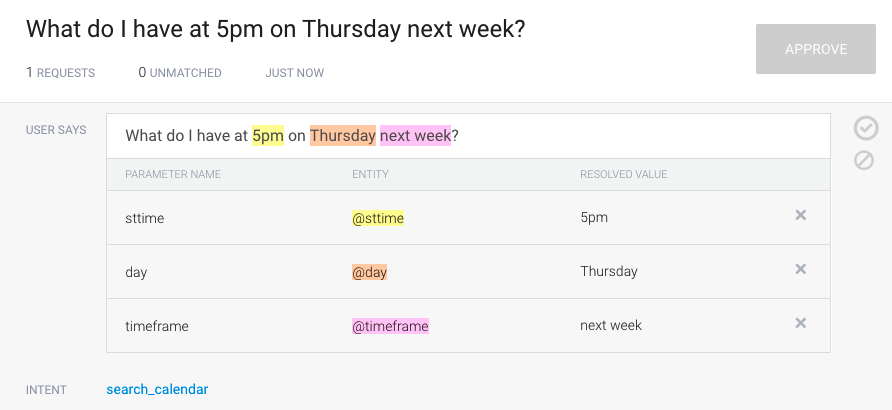
* + 1. Threads – primary key: thdid (thread ID)
       1. This tables is a simple map between thread ID’s and their corresponding users



* + 1. Events – primary key: thdid (thread ID)
       1. This table associates every thread with an event. The event may be already created or in the process of being created



**NLP**

1. API.AI guide
   1. How it works
      1. API.AI is an online NLP service owned by Google
      2. You can create an account and create ‘Agents’ to do tasks
      3. To access our agent, go to API.AI’s site and sign in with the ‘tycoagent@gmail.com’ email address and password ‘silvia2.0’
      4. You can train the agent on a current intent it already has on the UI provided by the site by typing in a sample sentence to the ‘Try it now’ text box in the upper right side of the screen
         1. API.AI will try to understand the sentence, and capture intent and entities
         2. It will show the intent and entities it currently thinks are in the sentence based on the current training
         3. If it is wrong, go to the Training section (left side of the screen), and look at the latest logged input, which should be what you just tested the agent on. You can change the intent and highlight or deselect parameters in order to fit what you want to extract from that sentence
         4. Then hit ‘Save’ or ‘Train’, and you should see the Settings Gear button spinning, which means that the Agent is now being trained on that example
      5. To add an intent, click the ‘Add Intent’ button on the left. Give the intent some example sentences with all the parameters possible for that intent.
         1. EX: If I wanted an intent called ‘weather’, and I wanted it to find the entities for ‘location’, ‘time’, ‘temperature’, and ‘type’, I would give it around 3-5 examples in the ‘Add user expression’ text box that have most if not all of the entities, such as ‘Is it raining hard in San Jose right now, and how cold will it be tonight?’. The location would be San Jose, the time would be tonight, the temperature would be cold, and the type would be raining
      6. To add an entity, click the ‘Add Entity’ button on the left.
         1. Give the entity different values. Ex: for an entity called vegetables, create different values like broccoli, carrots, etc
         2. Every value in an entity is resolved to one word, because there may be more than one way to say the same thing. Ex: Thursday is the resolved value for Thursday, Thurs, Thur, thursday, ... . This standardized the entity values, and the resolved value is what the parameter will show in the json
2. How to switch the NLP service
   1. All you have to do is kill the current rest server (api\_requester, which looks at API.AI), create a server that takes in the same type of json and outputs the same type of json, and connect that server to the NLP of your choice

**Email**

1. How it currently works
   1. Right now, we have an email\_server process that grabs emails every X amount of seconds, checks if they are from a registered user or part of a registered thread, then sends it to the tyco\_server in the form of a json Command object
   2. Email: [assistant.jci@gmail.com](mailto:assistant.jci@gmail.com)
   3. Pw: tycoagent2.0
2. How to add more interfaces
   1. To add more interfaces (Ex. Microsoft Outlook, Yahoo Mail), you need to set up a new email account for the service, and find their email API’s, and access them that way
   2. Then set up a server similar to email\_server that checks that account for new emails, then sends a json Command to the tyco\_server

**Calendar**

1. How it works right now
   1. We use the user’s refresh token to access their calendar, then based on the Command’s intent, we do some stuff on their calendar, like look for events, check to see if some time is free, delete events, or make events
2. How to add capabilities
   1. To add capabilities, the NLP service first has to be trained on the new capability you want to implement. Ex: We added the capability to search and delete to see if a certain time on a day was free, but we first needed to add sttime (time parameter) to the search and delete intents on API.AI so that can be extracted
   2. You then need to use that new data that will be extracted while utilizing the Calendar service
3. How to add more interfaces
   1. You need to find the API’s for the service you want to use (Ex: Microsoft Outlook Calendar, Yahoo Calendar), and use that
   2. Make sure the code you use to do stuff on that Calendar to do stuff utilizes the new and old extracted data

**Server/Client Setup**

1. Ports
   1. Tyco\_server has a Server Socket on the port 2003 (src/agentserver/TycoAgentServer.java)
   2. Api\_requester has a Server Socket on the port 3009 (src/apiai/API\_AI\_Requester.java)
2. Corresponding to the 2 servers, there are 2 ‘JSON-friendly’ objects: Command and Basic\_NLP\_Response
   1. Command (src/agentserver/Command.java)
      1. This has the following info in the object and in its json form
         1. Int type (type 0 means it is an email command => 1 mean it is an Alexa command)
         2. String (Alexa) alexaemail => the Amazon account email for an Alexa-registered user. We get their Calendar-linked email from this email address, which could be the same email
         3. String message => the message to be analyzed
         4. String from (Gmail) => the email address of the person who sent this particular email to the agent
         5. String subject (Gmail) => the subject line of the email
         6. String id (Gmail) => the message ID given by Google to this email
         7. String threadId (Gmail) => the thread Id given by Google to this email
         8. Long internalDate => the internal date the email was received by the agent in terms of UTC
         9. Map<String,String> recipients => a map of email addresses to names, if present, of all the people who received or sent this particular email
         10. Map<String,String> invitees => a map of email addresses to name, if present, of all the people who were ever a part of this email thread at any point
      2. The Command object is created in the email\_server from a Mail object and a String of the user’s email, then turned into a JSON and sent to the tyco\_server
      3. The Alexa lambda function writes a JSON that matches just ‘type’, ‘message’, and ‘message’
   2. Basic\_NLP\_Response (src/apiai/Basic\_NLP\_Response.java)
      1. This has the following info in the object and in its json form
         1. Boolean answered => true if the object has the intent and parameters for the message set
         2. Boolean failed => true if the request to the NLP failed at some point
         3. String intent => intent of the sentence set on the NLP server side
         4. Map<String,String> map of parameter name to value set on the NLP server side
         5. String sentence => the sentence to be sent to the NLP server to be analyzed, set on client (tyco\_server) side
      2. The Basic\_NLP\_Response object is created from just the sentence to parse in the tyco\_server flow, setting answered and failure to false. It is then sent as a json to the NLP server, which analyzes the sentence, then puts the intent and parameters in the object and sends it back
3. Server Threading
   1. The servers are set up in the following format
      1. Initialize server socket
      2. Run while(true) loop
         1. Try catch
            1. Check for a new connection to the server. If so, spawn a thread that deals with it and start waiting again (pass in the connected socket to the thread)
4. Starting the servers
   1. In order to start the 3 servers, you must run 3 jars on the AWS
      1. The first one is located at /home/ubuntu/TycoAgent/, and currently has the name ‘tyco\_server2.jar’
         1. This is the main backend server
      2. The second one is located at /home/ubuntu/TycoAgent/apiaiserver/, and currently has the name ‘api\_requester1.jar
         1. This is the NLP service server for API.AI
      3. The third one is located at /home/ubuntu/TycoAgent/emailserver/, and currently has the name ‘email\_server2.jar’
         1. This is the server that gathers emails and sends them to the backend
   2. I like to start the servers with the command ‘**nohup java –jar *jarnamehere*.jar &**’ while inside the directory they are located in
      1. What this does is start the jar as a background process while redirecting System print statements to a file in that directory called ‘nohup.out’
   3. Exact way to start the service
      1. First start the main backend server
         1. Go to directory /home/ubuntu/TycoAgent/
            1. cd /home/ubuntu/TycoAgent/
         2. Run the following command
            1. nohup java –jar tyco\_server.jar &
         3. Make sure that it ran successfully
            1. more nohup.out
            2. netstat –ntapl | grep 2003
      2. Then start the NLP Server
         1. Go to directory /home/ubuntu/TycoAgent/apiaiserver
            1. cd /home/ubuntu/TycoAgent/apiaiserver/
         2. Run the following command
            1. nohup java –jar api\_requester.jar &
         3. Make sure it ran successfully
            1. more nohup.out
            2. netstat –ntapl | grep 3009
      3. Now start the email server
         1. Go to directory /home/ubuntu/TycoAgent/emailserver
            1. cd /home/ubuntu/TycoAgent/emailserver
         2. Run the following command
            1. nohup java –jar email\_server.jar & (If you want to start checking for emails after right now)
            2. nohup java –jar email\_server.jar LONGTIME & (If you want to start checking for emails after the long UTC time LONGTIME)
         3. Make sure it ran successfully
            1. more nohup.out

**Alexa Setup**

1. Add Skill

a. Click on create new skill in amazon developers account

b. Fill out appropriate invocation name in the Skill information tab.

c. In the interaction model decide what intent schema to use

i. In our case, other than the default Amazon intents (Open, Close, Help), we added one custom intent rawinput, that solely was there to capture the raw speech text of whatever query the user says.

ii. In the json intent schema, our custom intent has a specified slot called “Item” that retrieves AMAZON.LITERAL which is the Alexa’s best guess of what the user is trying to say.

iii. This type of slot and custom intent is unique to us, because we are using Alexa solely as a speech to text interface and applying the NLP techniques at our own backend.

d. In configuration there is a two step process:

i. First you must link your endpoint (backend) with the alexa skill. For us this is the AWS Lambda Function we created and we simply just paste the function’s endpoint given to us by AWS once it was created. (More details on the lambda function later)

ii. Second allow Account Linking to occur on your alexa skill. This is what allows the alexa skill to know what user is currently using this skill, so it finds the appropriate calendar to search/delete in our backend. For the specific steps to link your account vist the website below: <https://developer.amazon.com/blogs/post/Tx3CX1ETRZZ2NPC/Alexa-Account-Linking-5-Steps-to-Seamlessly-Link-Your-Alexa-Skill-with-Login-wit>

e. Finally to wrap up, fill out the necessary information and upload the appropriate images on the Publishing Information tab to create preliminary app that user would see once they add the skill

1. Lambda Function on AWS

a. Lambda Function is the endpoint the Alexa first sends its data to. It can be in javascript, python or java. To begin you must create an amazon web services account and click on “create new lambda function.”

b. If your backend is in python or javascript you can directly copy and paste your code in the lambda function interface. In our case the backend is in java so we had to upload a jar that included our code and dependencies that we used.

c. Our java backend received the speech text the alexa captured and clarifies the intent the user has.

d. As soon as it clarifies the intent it send the appropriate response. If the intent is not the default open, close, or help it POSTs the query text data alexa received along with the amazon account that was linked when the user spoke to our PHP Client that is sitting on another AWS instance.

e. After POSTing the data the lambda function waits for a String response from the PHP Client which is what the alexa will respond back to the user.

f. Once you upload your backend code to lambda function and save the function on the top right corner of your screen there will be your function id. This id is what you will copy and paste in the alexa configuration tab when you are creating the skill and linking your endpoint.

g. Samples of good java lambda functions can be found at <https://github.com/amzn/alexa-skills-kit-java>

1. PHP Client

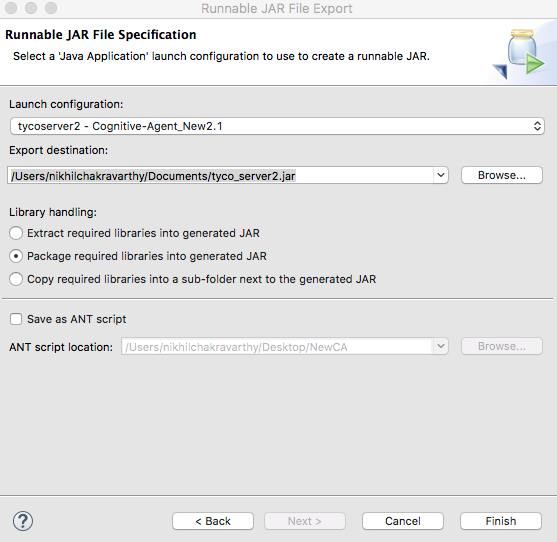
a. Our PHP Client is the middleman that receives the necessary information from the Alexa lambda function and relays the important information to our own backend in a way that our backend can understand.

b. After receiving the query and account info from the POST the php encodes the important text data in a json with the additional information that this is an Alexa query (by including that this query is type 1).

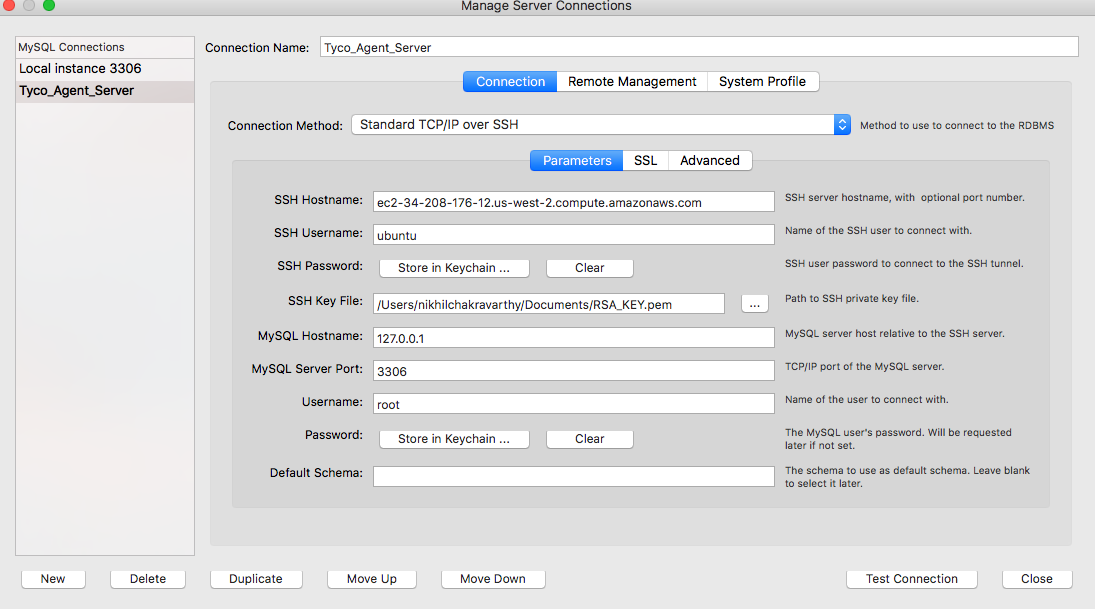
c. As a client it opens a socket on port 2003 and sends json object to server that is listening to the same port in our backend. The backend then does the NLP analysis and the appropriate actions associated with the request. Finally the backend constructs the message the user should receive after the request has been processed and it sends the response back to the client on port 2003.

d. The PHP Client is waiting for the backend to finish processing the query and when it finally receives a response string data from the server it immediately closes socket and send string response back to lambda function which will ultimately send it to the alexa.

**Demo Setup**

1. Testing Gmail/Amazon accounts (preauthorized with the agent with Amazon email linked
   1. first Gmail/Amazon account: [tester.1.jciagent@gmail.com](mailto:tester.1.jciagent@gmail.com)
      1. password for everything: pizza\_antica
      2. bday: UTC start time, Jan 1, 1970
   2. seconds Gmail/Amazon account: tester.2.jciagent@gmail.com
      1. password for everything: eriks\_deli
      2. bday: UTC start time, Jan 1, 1970
2. Testing Alexa (preauthorized with the agent with Amazon email linked
   1. Have an echo that is linked to the amazon account [tester.1.jciagent@gmail.com](mailto:tester.1.jciagent@gmail.com)
   2. enable skill on the [tester.1.jciagent@gmail.com](mailto:tester.1.jciagent@gmail.com) alexa home
   3. Sign into the app with the user that you want to search/delete calendar events from (make sure this user has previously in preferences indicated his amazon email and said yes to alexa integration)
   4. Then open JCI agent and send your request.
3. Authorize via Website
   1. In order to authorize a Gmail to be signed up for JCI Agent, go to http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com/apache/index.html, the websites homepage
      1. Then go to the Sign-Up page and sign up with the email you want to let the agent have calendar access to
      2. Give it a password, and give your Amazon account email if you would like to also have Alexa integration
      3. Set preferences if you would like
4. Get source code
   1. The source code is located on the AWS <http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com/> at home/ubuntu/TycoAgenthttps://github.com/ashoksundaresan/Cognitive-Agent\_New
   2. Use this command when you are in the directory you want to download the project into
      1. git clone https://github.com/ashoksundaresan/Cognitive-Agent\_New.git
5. Dependencies
   1. With the source code, there should be a pom.xml file. It has all the dependencies you need to implement the source code and use the various API’s
   2. In order to to get the dependencies on the computer you are work on, download the apache maven 3.5 bundle
   3. Then put the pom in an empty directory, then run the command ‘mvn clean install’ while in that directory (the file needs to have the name ‘pom.xml’). This will have maven go into its repository and grab the needed jars, putting them under the folder ‘~/.m2/’
6. Building the Runnable Jars
   1. After getting the source code and the dependencies, open the project in Eclipse
   2. After making the changes to the source code, save the project
   3. Now to export the project as 3 separate runnable jars
      1. Tyco\_server.jar
         1. File > Export > Runnable JAR file
         2. Select the correct launch configuration that corresponds to running the main method in TycoAgentServer
         3. Select the file destination, and click finish
      2. Api\_Requester.jar/Emailserver.jar
         1. Follow the same steps as for tyco\_server.jar, but give the jars a different name and select the correct launch configuration for each
            1. Api\_requester’s configuration should be the main method in the API\_AI\_Requester class
            2. Email\_server’s configuration should be the main method in the EmailServer class
   4. Now you have to push the JARS to the AWS at <http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com/>
      1. Use the Secure Copy command with the path to the keyfile to put each jar under their correct spot in the AWS
         1. /home/ubuntu/TycoAgent/tyco\_server.jar
         2. /home/ubuntu/TycoAgent/apiaiserver/api\_requester.jar
         3. /home/ubuntu/TycoAgent/emailserver/email\_server.jar
      2. Example:
         1. scp -i /Users/nikhilchakravarthy/Documents/RSA\_KEY.pem /Users/nikhilchakravarthy/Documents/tyco\_server2.jar [ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com:~/TycoAgent/](mailto:ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com:~/TycoAgent/)
            1. \*\*NOTE\*\* ‘~’ means the home directory, /home/ubuntu/
7. MySQL Database Setup
   1. Installation
      1. Go to <https://dev.mysql.com/downloads/mysql/> and select the type of MySQL service you want (we currently use the Community edition)
      2. To download onto an AWS from command line, use this website <https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-16-04>
      3. NOTE\* MySQL’s default port is 3306
   2. 2 options
      1. Create a port forwarding from your current computer’s port 3306 to the port 3306 of the [ec2-34-208-176-12.us-west-2.compute.amazonaws.com](http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com/) AWS
         1. ssh -i /Applications/XAMPP/xamppfiles/htdocs/apache/SSH/tyco\_garage\_us2.pem -L 3306:127.0.0.1:3306 ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com
      2. Create a MySQL instance with the same Schema name, Table names, and Table setup that operates on port 3306 of the computer that will be running your version of the Agent. However, if you do this, you will have to manually put in User information. I suggest the first option
8. Initialization
   1. There are 3 files that should be run. I did this by compiling the project into 3 runnable jars that each run one of the files’ main method
      1. The first file to be run is the TycoAgentServer, located in src/agentbackend/agentserver/
      2. The second file to be run is the API\_AI\_Requester, located in src/agentbackend.responsegeneration/apiai/
      3. The last file to be run is the EmailServer, located in src/email/server/
         1. This file takes in either 1 or no inputs. If and input is given, it should be a long form of UTC (milliseconds since Jan 1, 1970). If no input is given, the default is the current UTC. This is the time the program will check for emails after for the first email check
         2. When this program is run for the first time on a new computer, it should be run via an IDE or from command line without redirecting the output away from the command line. This is because you need to authorize the program to have access to [assistant.jci@gmail.com](mailto:assistant.jci@gmail.com). Just run the program and follow the instructions that are printed and in the browser window that will open up. A Java Stored Credential object will be stored at ‘~/.credentials/tyco-revamp/’ that will bypass the need to always manually authorize the program after the first time

**FAQ**

1. Current capabilities
   1. The agent can do search and delete for Alexa, and all 4 (search, delete, insert, update) for email
      1. Search
         1. The agent can search by day, this/next week, date, and time
      2. Delete
         1. The agent can delete by day, this/next week, date, and time
      3. Insert
         1. The agent can recognize days, dates, times and this/next week to set up events. Locations, however, has the limitation to just ‘lobby’, ‘Starbucks’, and ‘Chipotle’. A CSV of locations should be uploaded to the location entity (tutorial on API.AI’s site)
      4. Update
         1. Can only update events set up by the agent by responding to the thread that the event was set up on
         2. Same entity recognition as Insert
2. Next steps
   1. Add more mail and calendar interfaces/connectors
   2. Have insert and update work on Alexa
   3. Search and delete by name, location, other info, etc.
   4. Check that everyone on the thread is ok with the times for the meeting before creating it
   5. Create domain name for website
   6. Organize website files, make website more conventional
   7. Scale out the agent to more processing power than just the AWS it is hosted on
   8. Ensure that mail received from non-Gmail providers can be handled properly
   9. Handle misspellings of keywords (entities; ex: Thurday vs. Thursday)
3. Current Scalability
   1. The agent server is well threaded, so the limit to scalability currently is the AWS’s processing power/memory
   2. In my best guess, the program should be able to handle traffic from the entire Tyco Innovation Garage team plus some Alexa users easily
4. Areas to test
   1. Sentence parsing
      1. In Pre\_NLP\_Parser.java, we use a little machine learning from Open NLP to smartly split messages over 256 character into a list of sentences, which we then send to API.AI sentence by sentence, then compile each sentence result into a comprehensive Basic\_NLP\_Response. This needs to be tested more, since we only did one proof-of-concept test
   2. Update
      1. We have not used update since early prototyping, and just generally testing to see if events are being properly updated
   3. Emails from different services
      1. When [assistant.jci@gmail.com](mailto:assistant.jci@gmail.com) receives emails from a Yahoo or Outlook mail account, sometimes the emails don’t get parsed correctly and some things may happen
         1. the email is not recognized, so an error is thrown
         2. the total email content can’t be properly trimmed of the auto-created timestamps, causing an error in NLP analysis
         3. We get lucky at it works
      2. This is also an area to expand in
   4. NLP
      1. Test API.AI, Open NLP, and other NLP services to see which one get’s the best results
      2. Currently, we have tested API.AI and Open NLP, and API.AI is better
5. Where is the current deployment
   1. AWS located at <http://ec2-34-208-176-12.us-west-2.compute.amazonaws.com>
   2. To log in, use SSH with the approprient .pem file (contact Ashok Sunderesan for the .pem file)
      1. ssh -i /Applications/XAMPP/xamppfiles/htdocs/apache/SSH/tyco\_garage\_us2.pem [ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com](mailto:ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com)
      2. If you would like port forwarding:
         1. ssh –i /Applications/XAMPP/xamppfiles/htdocs/apache/SSH/tyco\_garage\_us2.pem -L 3306:127.0.0.1:3306 [ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com](mailto:ubuntu@ec2-34-208-176-12.us-west-2.compute.amazonaws.com)
         2. This allows the computer you are using to access the MySQL Database on the AWS directly from its 3306 port, if it is not being used
   3. Apache server located at /var/www/html/apache
      1. Under this, the entire website (html, php, css, etc) is located
   4. Backend server
      1. /home/ubuntu/TycoAgent
      2. To start the servers, refer above
   5. MySQL Server
      1. Installation above
      2. Download MySQL Workbench here: https://dev.mysql.com/downloads/workbench/
      3. To set up tables and schema, use MySQL Workbench from the computer you are using, and go to the SSH option
   6. Logging
      1. Logfiles for each of the servers are located in the same director as the jarfile, names ‘nohup.out’

Source code located here: <https://github.com/ashoksundaresan/Cognitive-Agent_New>

Thank you

Written by Nikhil Chakravarthy and Nihal Maunder