

Synnex Fact-Checking Portal

Andrew Padgett, Caleb Grillo, Harrison Hall, Kaleb Spraker



Contents

Contents	1
Project Description	2
Description	2
Vision	2
SYNNEX	2
Team	3
Andrew Padgett	3
Harrison Hall	3
Kaleb Spraker	3
Caleb Grillo	3
Project Scope and Schedule	4
Scope	4
Schedule	5
User Guide	7
Front Page	7
Interacting with Facy	8
Account System	10
Admin Controls	11
Developer Guide	12
Stack	12
Algorithm	13
UML Diagrams	14
Facy Algorithm	14
Services Overview	15
Development	16
Repository and Future Development	16
Endpoints	16
API Keys	17
Challenges	18
Follow-up Questions	18
Active Learning	18
Fact-Checking with LUIS	18
Parsing Information with Indexers	19
End Result	20

Project Description

Description

We pitched a web application that would answer user questions and fact-check user assertions on the topics of Clemson sports and the Battle of the Alamo. The web application would allow users to sign in, upload factual documents, and track queries through the portal. We wrote the bot with a custom algorithm combining technologies like Azure QnA bot and Azure LUIS. SYNnex wanted the focus of the Capstone project to be on developing an interactive AI solution that harnesses the power of Microsoft Azure to solve problems.

Vision

SYNNEX wanted us to create a derivative of the last project students' made for SYNnex in the capstone class. This means using Azure's Chatbot Framework and Cognitive Search to create a bot with the theme of "fact-checking." SYNnex wanted the students' vision to form what the project would be.

Our vision was to create a web application that allows users to interact with a chatbot and designated admin users to alter Factly's knowledge base or its brain. The chatbot would answer questions and fact check statements that relate to the information held in its knowledge base. In case of outdated or flawed information, admin users can delete and upload documents Factly pulls from directly from the account page.

SYNNEX

SYNNEX Corporation provides business-to-business IT services that help partners grow and enhance their customer engagement strategies and is an industry leader in IT hardware distribution and customer care software solutions. Synnex has operations in more than 30 countries and has over 100,000 employees. Having gone public on the New York Stock Exchange in 2003, SYNnex is currently ranked 169 on the 'Fortune 500', with over 18 Billion in annual sales. SYNnex brings the most relevant technology solutions to the IT and consumer electronics markets by distributing more than 30,000 technology products and services from more than 300 of the world's leading and emerging manufacturers (including Microsoft, Google, RedHat, etc.).

Team

Andrew Padgett

Andrew was the team lead and managed most correspondence between the group and SYNEX. He worked with Harrison to develop the flask web app and designed most of the frontend components. Andrew worked with Kaleb to construct the algorithm to train the chatbot through user feedback and helped Kaleb figure out how to parse tables within text documents.

Harrison Hall

Harrison worked closely with Andrew on the backend of the website. He helped connect the backend to the frontend by writing the algorithm used to respond to user queries. He helped develop the database system and was responsible for user accounts, query tracking, and document logging. Working with Caleb, Harrison helped implement the usage of LUIS on the backend to fact-check assertions through the QnA bot.

Kaleb Spraker

Kaleb Spraker worked on the backend of the website and with the resources in Azure Portal. His focus was mainly on Cognitive Search and Knowledge Base updates for the QnA bot. He also created the Azure Functions that parse and make Question and Answer pairs from documents located in Blob storage on the Azure Portal. Along with the functions, Kaleb also created the indexers and custom skill sets that use those Azure Functions to parse documents.

Caleb Grillo

Caleb worked with the Azure side of the project, familiarizing himself and utilizing the resources Microsoft has to offer. He was given the tasks of learning the different frameworks for Bing search, QNA bot, Video Indexing, and LUIS and finding beneficial ways to use these in the scope of the fact-checking bot.

Project Scope and Schedule

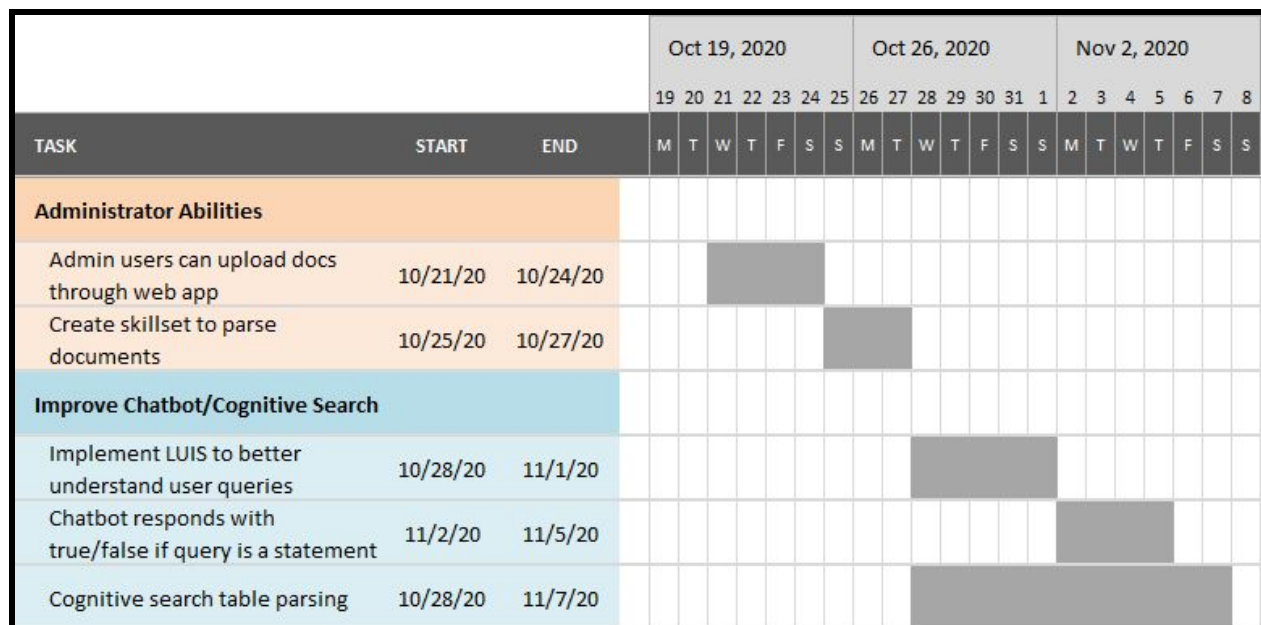
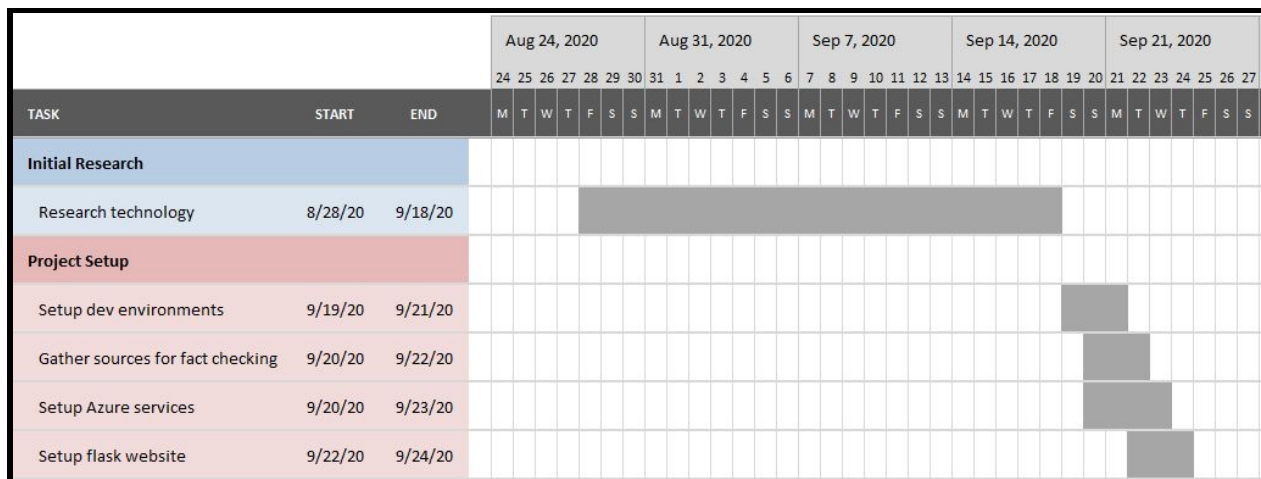
Scope

We had the whole Fall semester (starting August 28th, 2020) to create a Fact-Checking Chatbot about any subject of the team's choosing. The Chatbot should be able to ingest different documents and learn the information in them. It should also be natural in language and conversational. The webpage should have some exciting and engaging features besides just the chat. Trusted users can upload and delete documents relating to the topics specified.

Going more in-depth, the Chatbot checks facts and answers questions about Clemson Football, focusing on Quarterback Trevor Lawrence. Also, the Chatbot's focus includes information about the Battle of the Alamo. It learns and remembers information in word and pdf documents, including the tables in pdf documents. The Chatbot learns from its interactions with users and should train itself from its conversations, meaning it can only get smarter. If anyone ever asks the chatbot something it does not have an answer for, it will return the top result from bing.

Schedule

The Gantt chart below shows our final timetable and what features we spent time on throughout the semester. The chart is split up to keep the text from being too small to read.



			Sep 21, 2020							Sep 28, 2020					Oct 5, 2020							Oct 12, 2020							Oct 19, 2020							Oct 26, 2020										
			21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1		
TASK	START	END	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S		
Chatbot Implementation/Flask Web App																																														
Chat QnA/Realistic Response	9/25/20	10/1/20																																												
Test bot	9/30/20	10/5/20																																												
Implement chatbot with web interface	10/4/20	10/8/20																																												
Setup database	9/28/20	10/2/20																																												
Account system	10/2/20	10/7/20																																												
Save/view queries	10/7/20	10/11/20																																												
Cognitive Search/Bing Search																																														
Setup blob storage for documents	10/12/20	10/13/20																																												
Create skillset to parse documents	10/14/20	10/15/20																																												
Customize skillset	10/16/20	10/30/20																																												
Setup Bing search	10/12/20	10/14/20																																												
Perform Bing search when no answer found	10/15/20	10/19/20																																												

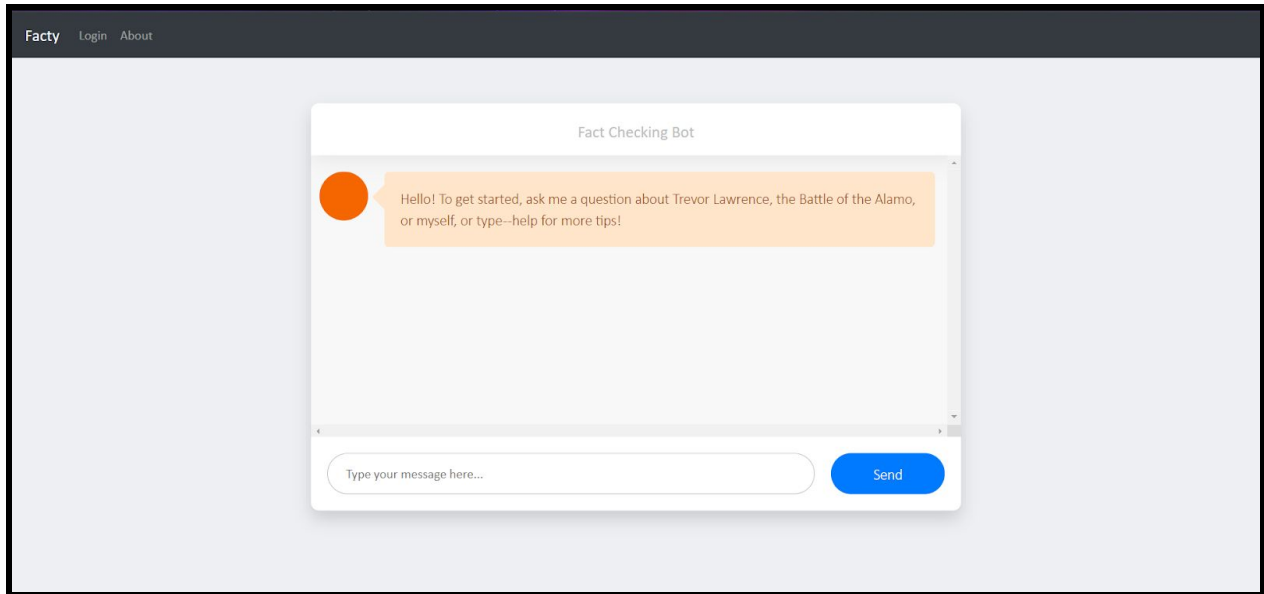
			Nov 9, 2020							Nov 16, 2020							Nov 23, 2020							Nov 30, 2020						Dec 7, 2020									
			9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13		
TASK	START	END	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S		
Project Refinement																																							
Make interface easier to view	11/9/20	11/19/20																																					
Tweak LUIS configuration	11/9/20	11/16/20																																					
Add more documents for bot to pull from	11/20/20	11/23/20																																					
Test chatbot	11/20/20	11/25/20																																					
Build final report	11/20/20	12/8/20																																					

User Guide

Step 1 is simply navigating to the webpage that facty lives on; here is the URL:
<https://fact-checker.azurewebsites.net/>.

Front Page

Once you've successfully navigated to the webpage, you should see something like this:

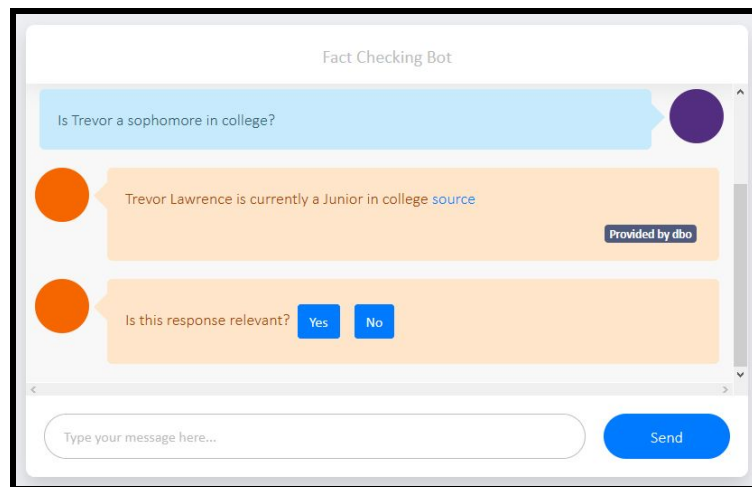


You'll see several interactions available to you. The chatbot takes center stage on the webpage, and there are two buttons located at the top left of the screen: Facty, Login, and About.

Interacting with Facy

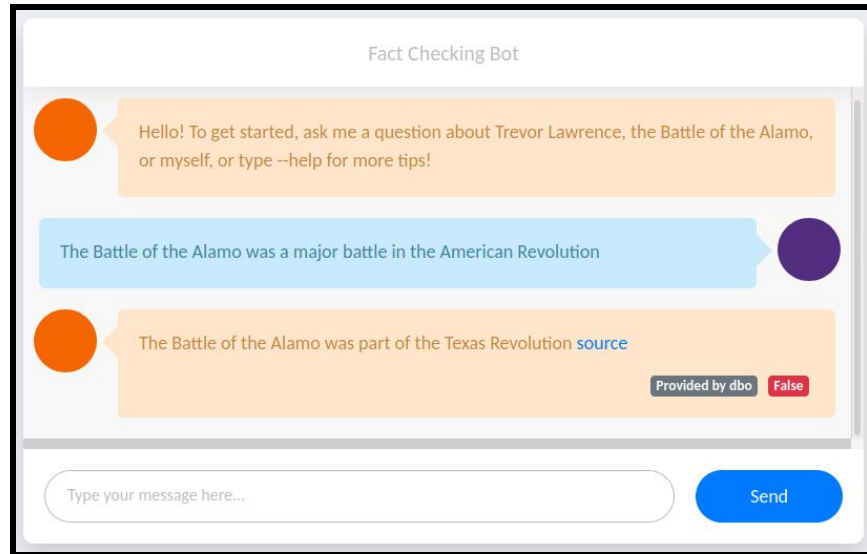
To interact with Facy, ask it any question or give it a statement related to the information Facy knows. When asking a question, three different results can happen:

1. Facy simply answers your question.
2. Facy answers your question and asks if the answer is relevant to your question (shown on the next page). This happens when Facy is not highly confident in its response.



- a. Click **"Yes"** to tell Facy that the answer given is answering your question. Facy will remember this later.
 - b. Click **"No"** to tell Facy its response does not correspond to your question, and Facy will try again before resorting to Bing.
3. Facy has no idea how to answer your question and will ask Bing for an answer.

When giving Facy a statement to fact-check, facy will respond with the answer, and in the bottom right corner of the response, you will see a flag, saying if what you stated was **"True"** or **"False."**



(Here you see that Factly responded with False)

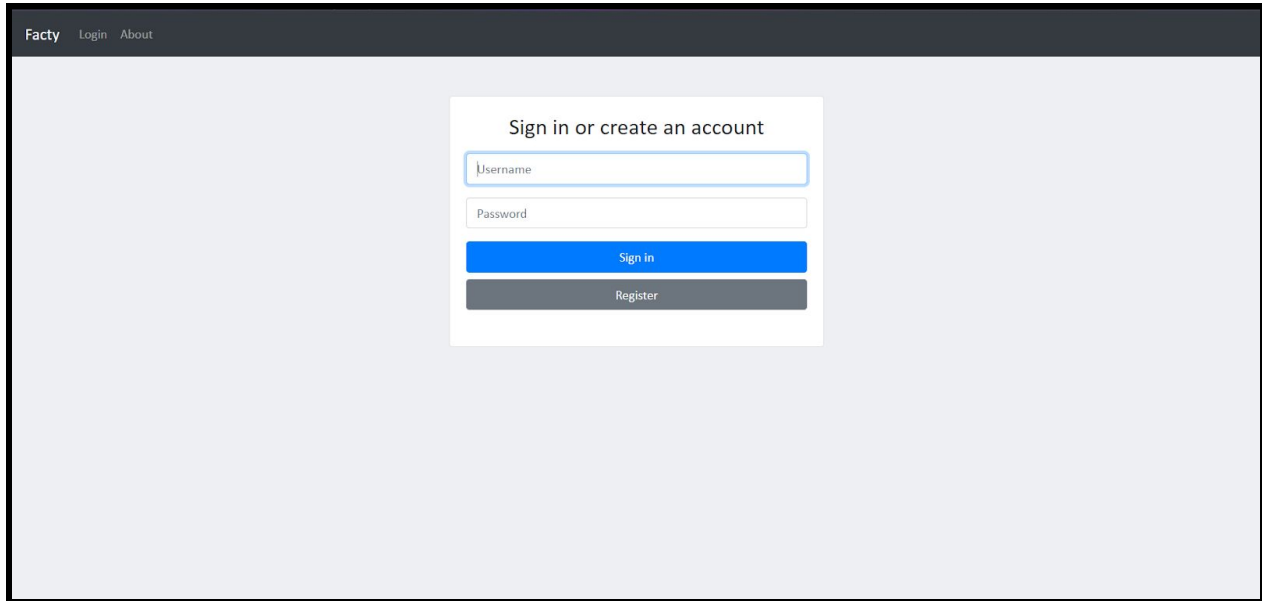
From the bar at the top of the page, selecting **"Facty"** will bring you back to the main chatbot page from wherever you are. Choosing it on the Factly page will simply just reload the page.

Selecting **"About"** navigates you to the "About Us" page, offering a brief overview of what facty is and who developed it.

To get back to the main page, simply select **"Facty"** in the top left, as mentioned previously.

Account System

Here we'll talk about how users can make an account and what that account offers them. Selecting **"Login"** in the top left of the page will bring the user to a login page:



The screenshot shows a web application interface. At the top, a dark gray navigation bar contains the text 'Facy' followed by links 'Login' and 'About'. The main body of the page is light gray. Centered in the middle is a white rectangular box with the title 'Sign in or create an account'. Inside this box, there are two input fields: 'Username' and 'Password'. Below these fields are two buttons: a blue button labeled 'Sign in' and a gray button labeled 'Register'.

Following the on-screen instructions, enter a username and password. Assuming you don't have an account, click the gray **"Register"** button at the bottom of the box.

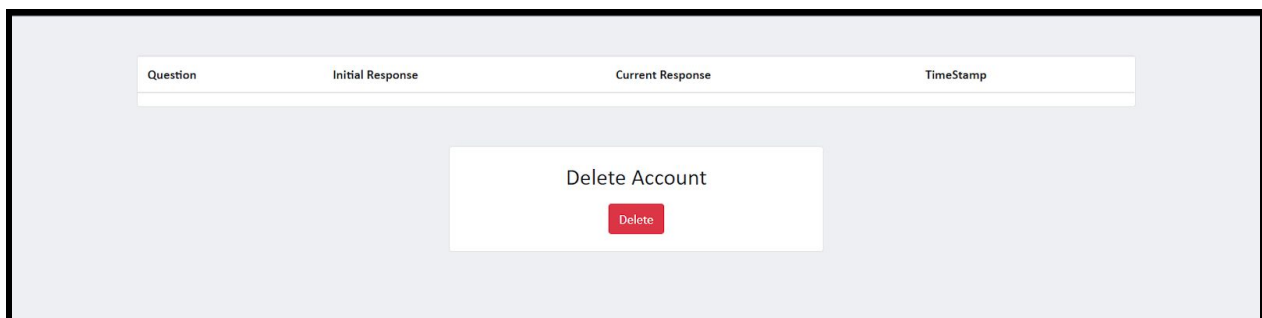
You will be redirected to the main page and notice your top bar now includes your **"Username"** in the top left and **"Logout"** in the top right.



This screenshot shows the top navigation bar after a user has logged in. The dark gray bar now displays 'Facy' followed by 'YourUsername' and 'About'. The 'Logout' link has been moved to the right side of the bar.

Selecting Logout will simply log you out of your account.

If you already have an account, select the blue **"Sign In"** button to sign in to your account. This redirects you back to the main page and changes the top bar to look the same as above. Selecting your Username in the top left will bring you to an account page, where you can view past questions you have asked Facy and a **"Delete Account"** button.



The screenshot displays an account management page. At the top, there is a table with four columns: 'Question', 'Initial Response', 'Current Response', and 'TimeStamp'. Below the table, centered on the page, is a white box with the title 'Delete Account'. Inside this box is a red button labeled 'Delete'.

Selecting the big, red **“Delete”** button at the bottom of the page prompts an insurance message, to make sure you want to delete your account. Selecting yes will delete your account, and it is not recoverable, so only do it if you wish.

Admin Controls

Assuming your account type is an admin, you have a few more interactions and abilities. Navigating to your account page as described previously, you will notice that you have this new selection at the top of the page:

Upload new File

No file chosen

Selecting “Choose File” will allow you to choose a document (only docx and pdf files) from your file system to upload into Facy’s brain.

Further down the page, you will see this table:

Document Name	Container	TimeStamp		
Alamo.docx	docs	2020-12-01 18:09:35+00:00	<input type="button" value="Download"/>	<input type="button" value="Delete"/>
TrevorLawrenceStuff.docx	docs	2020-12-01 21:05:43+00:00	<input type="button" value="Download"/>	<input type="button" value="Delete"/>
Trevor_2018_Article.docx	docs	2020-11-12 19:13:19+00:00	<input type="button" value="Download"/>	<input type="button" value="Delete"/>
clemsondates.docx	docs	2020-11-05 18:42:36+00:00	<input type="button" value="Download"/>	<input type="button" value="Delete"/>
clemonstuff.docx	docs	2020-11-05 18:43:19+00:00	<input type="button" value="Download"/>	<input type="button" value="Delete"/>

This is just a table of the current documents in Facy’s brain.

Selecting the blue **“Download”** button will download that document into your file system for your viewing.

Selecting the red **“Delete”** button will delete that document and any information relating to that document from Facy. Only do this if you are sure it is necessary.

Developer Guide

Stack

The Factly web portal is written as a Flask application and is hosted on Microsoft Azure. Flask is a python-based web server framework for creating web applications.

Azure

Functions

Azure Functions update our Knowledge Base as information is parsed from our dynamic Indexers.

LUIS

LUIS was used to break sentences up into their corresponding entities, key phrases, and datetimes. We were then able to compare these entities to see if two assertions agreed or disagreed. This was the basis for our fact-checking algorithm.

QnA Service

The Azure QnA Service was used to hold key-value pairs for responding to questions and assertions.

SQL Database

The Azure SQL Database held user information, queries, documents, and other persistent miscellaneous information.

Web App

Azure's web app service hosts our flask application on a Linux machine.

Bing Web Search

For QnAbot responses with low confidence, we utilized the Bing Web Search service to answer a question.

Website

Flask

The capstone team used Python's Flask framework for creating the web application server.

Jinja

Jinja was utilized to template and fill the HTML of the website. Jinja allowed the team to dynamically make web pages based on user account information and alter endpoints based on administration privileges.

Cryptography

The python cryptography library was used to hash and compare passwords stored in the database's Users table.

Bootstrap & jQuery

Bootstrap was used to format the website. Bootstrap was also utilized to create advanced functionality such as modals for confirming document deletion. JQuery is a dependency for Bootstrap, but it was also used independently to manipulate the DOM conveniently.

Showdown

Showdown helped format the responses from the QnA bot that were formatted in markdown syntax.

Algorithm

A user interacts with Factly by going to our website and typing to the bot. The message, along with a couple of flags, is sent to our Azure App Service Server. Based on the flags, the server will either create a custom response, answer a posed question, or fact-check a sent assertion.

If the query was a question, we send the question to our Azure QnA bot to reference the growing Azure Knowledge Base. If the QnAbot's response is confident enough, we send it back to the user. If the confidence is marginal, we ask the user if the response is relevant, teaching the bot based on the response. If the QnAbot is very unconfident, we utilize Bing to find the next best answer.

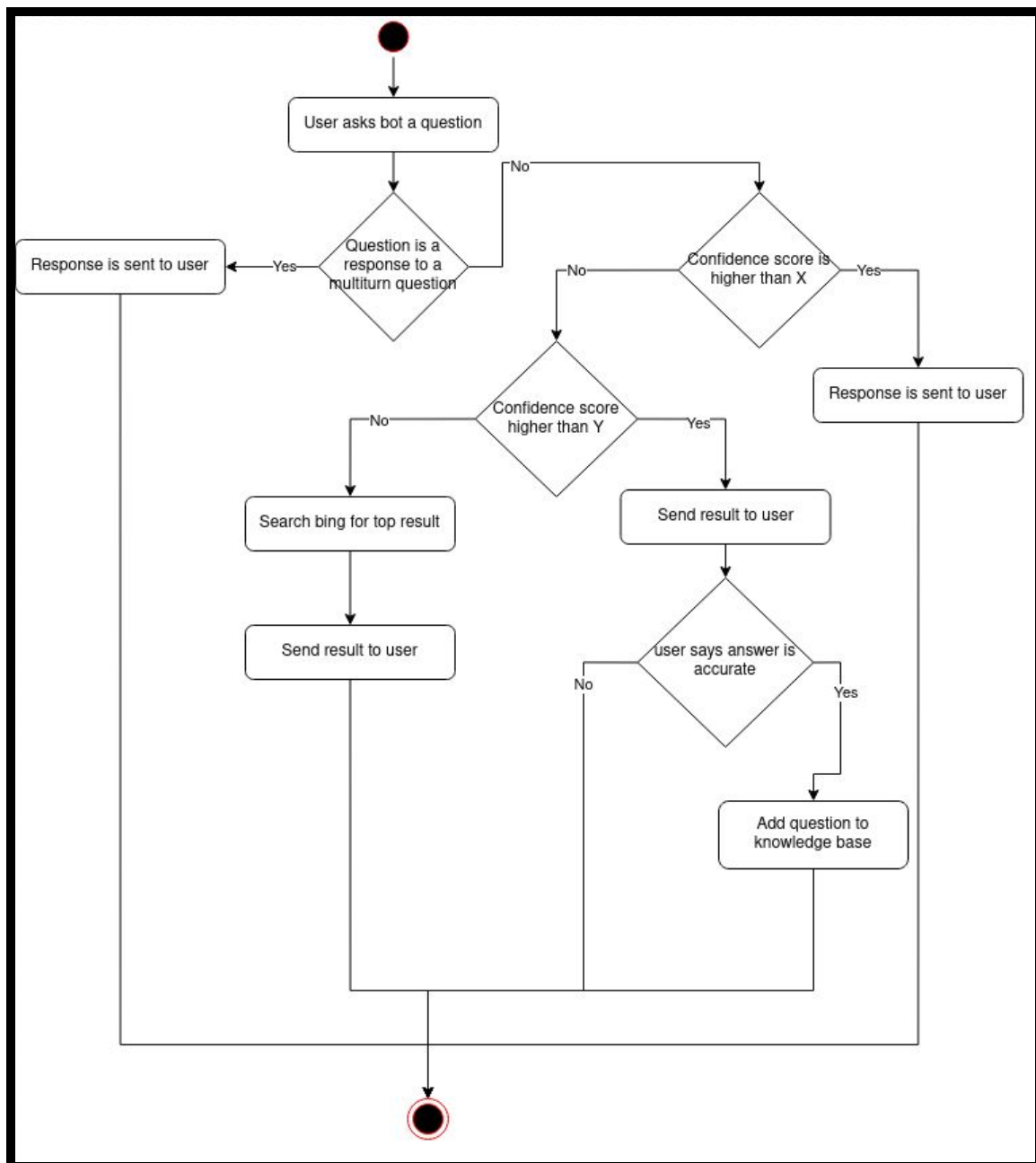
Similarly, if the query was an assertion, we still look for a response from the QnAbot. We send both the query and the response to LUIS to find out what intent and entities the statements

have. Using our fact-checking algorithm, we get a value comparing the two statements. That value, and the answer, are sent back to the user to flag the assertion as true or false.

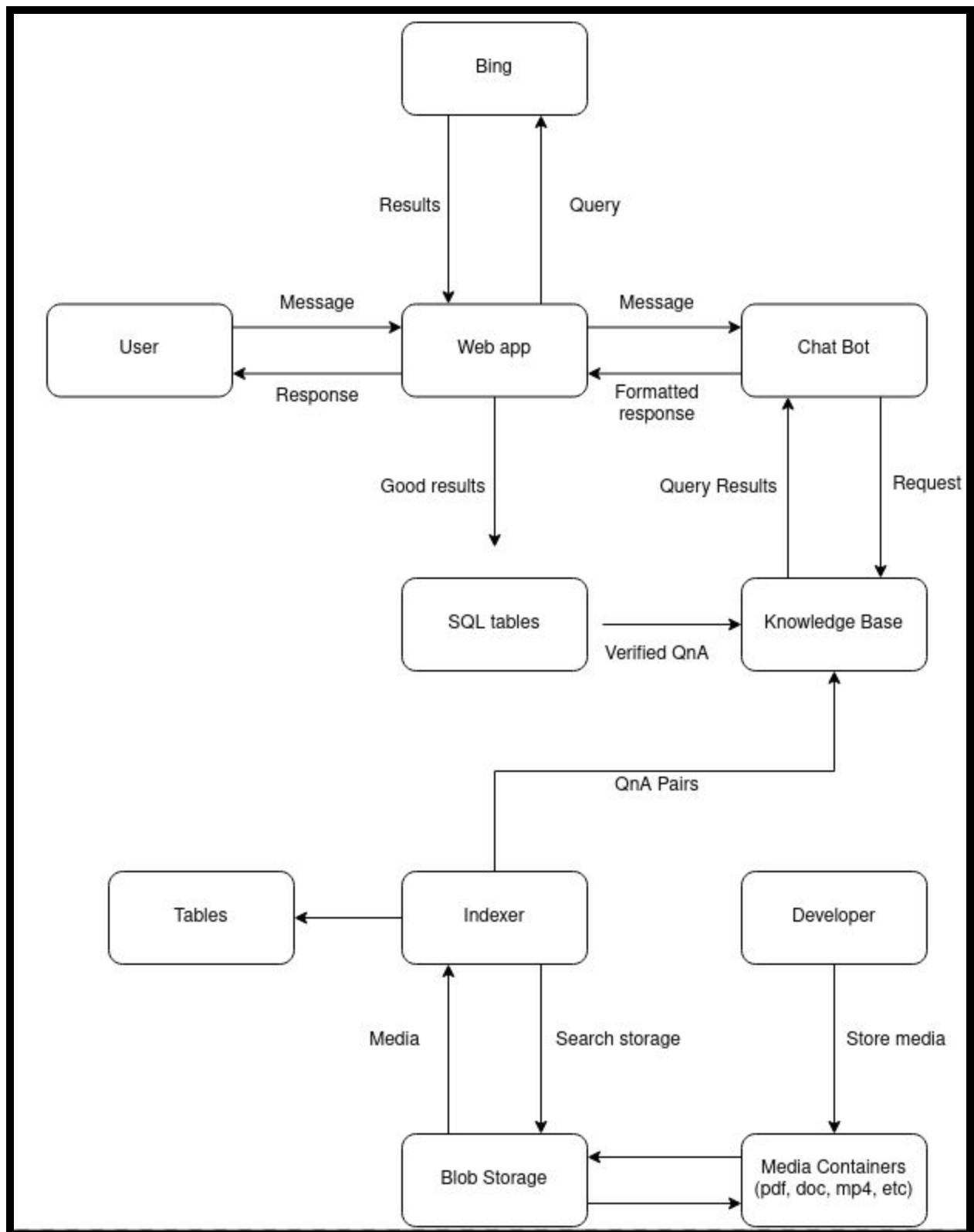
All activity is logged in our Azure SQL database so the bot can learn and users can reference what they have done previously.

UML Diagrams

Facty Algorithm



Services Overview



Development

The original algorithm simply acted as a middle-man between the Facy QnAbot and the user to secure our API keys. When we implemented active learning, we created our own form of active learning to help the bot learn. This required us to get user feedback from the bot. We created a new system of flags to handle this. When we decided to differentiate user input into questions and assertions, we rewrote this to consult LUIS if the user input was an assertion.

Repository and Future Development

The code repository holds the code for the web application, templates, stylesheets, and auxiliary scripts for working with Azure. ``app.sh`` and ``powershell.sh`` contains the script to execute the web app on Linux and Windows machines, respectively.

``scripts`` contains test and setup scripts for managing Facy. ``scripts/database_setup.sql`` contains the SQL code necessary to create the SQL tables required for the app.

``src`` contains the python modules used by the application. Each module has a header that describes the general purpose of the functions it contains. ``static`` contains the static web elements: the CSS, javascript, and images of the website.

``templates`` stores broken down HTML elements following the jinja2 templating system.

``templates/cards`` holds the filled Bootstrap cards that our website is broken down into.

``templates/components`` contains other components that fill our webpage. ``templates/pages`` holds the pages that are made by combining the cards and components.

Development of the system can be easily understood by reading the README ``README.md`` and pushing your changes to the azure web app. ``az webapp up -n fact-checker`` will push the current commit of the repository on Azure. If issues persist on the internet and not on a local version of Facy, debugging can be completed using the Azure portal.

Endpoints

Endpoints for the web app are defined in ``application.py``. Endpoints correspond to the different webpages the application serves and locations the user can request information from.

The root endpoint ``/`` returns the webpage for the index of the website. This corresponds to the `index.html` page. This gives the user the ability to communicate with Facy.

The about endpoint ``/about`` renders the webpage for our about page, which describes the Facy project.

The login endpoint `"/login"` renders the webpage for logging in to our system. The login card on the login page also allows users to create an account. Account logins are persistent due to cookies given by the Flask Session module.

The account endpoint `"/account"` renders the web page for accounts. This is only accessible when a user is logged in. Administrative users have access to cards with the features of uploading, downloading, and deleting documents. All users have access to the card to view past queries.

The account deletion endpoint `"/deleteaccount"` deletes the account in the user's session. This endpoint is only available if a user is logged in.

The logout endpoints `"/logoutbutton"` and `"/logoutrequest"` removes the account login information from a user's session. This endpoint is only available if a user is logged in.

The QnA endpoint `"/qnarequest"` allows the user to interact with the Facy algorithm. This is what the Facy chatbot window uses.

Document endpoints `"/upload"`, `"/upload_url"`, `"/delete_document"`, and `"/download_document"` allow the user to manipulate the documents that Facy is composed of. These are only accessible to administrative users.

API Keys

The Bing API key is located in ``src/bing.py``. This key corresponds to the Bing service, which Facy consults to find the answer to unknown queries.

The database key is located in ``src/database.py``. The functions associated with the database handle user data, administrative privileges, and persistent data.

The blob storage keys are located in their respective functions in ``src/documents.py``. This handles uploading and downloading documents from Facy.

The LUIS key is located in ``src/luis.py``. This handles breaking apart sentences using the LUIS rest API.

The QnABot key is located in ``src/qna.py``. This handles getting responses from Azure's QnABot service. It can also be used to update the corresponding knowledgebase.

Challenges

Follow-up Questions

Since our web app design is stateless, we needed an efficient way to track what a user responds to. Questions and assertions should be handled one way, while answers to follow-up questions should be handled in another.

To differentiate our traffic, we created a system of flags to know what response and responseID a query may be responding to, along with a variable for knowing which custom response it might be referring to. We utilized this approach to handle when follow-up questions are asked, allowing a list of questions to follow a single response. This allowed us to implement the Facy algorithm easily as it was developed throughout the semester.

Active Learning

The Azure chatbot's active learning feature is more suited for heavy traffic, meaning that it needs a lot of user interaction to improve its responses. Since a student project does not receive much user traffic, we decided to create our own learning system. Any time Facy is not very sure of its response, it will ask for user feedback on how relevant its response is to the user's question. If the user clicks yes, Facy will store the user's question in its knowledge base and pair it up with the answer it gave. Now every time a user asks that question, Facy can be sure of its response.

Fact-Checking with LUIS

Once we had the structure of our system under our control, we realized that we needed to immediately begin developing the fact-checking ability. When broken down, this requires us to compare the user's assertion and the best response given by our QnA bot. Ideally, this system would provide us with a value between 0 and 1, corresponding with how similar the two statements are.

We were recommended to look into Microsoft Azure's LUIS, a service that can find a sentence's intent and split the sentence up into its corresponding entities. A simplified example on the right shows how LUIS might break down the query "Trevor Lawrence ran 3 touchdowns in 2019." Since human language is complex, we needed a safe way to compare the information that the two statements held. After we learned how to utilize it and trained it on a host of different sentences, we wrote an algorithm that would compare each sentence's entities and return the

best-case value for comparing all of the entities. It calculated the percent difference of dates, Levenshtein distance of names and places, and similarity between statistics. While this solution has weaknesses, such as not being able to factor in negations or account for acronyms, we felt that this was a very good solution to a challenging problem.

Parsing Information with Indexers

We've mentioned how facty needs to be fed documents to fill its brain, well you can think of indexers as how facty reads and studies those documents. Finding complete sentences in documents was a little difficult in the beginning. There were some services offered to us that found sentences and information in the documents, but they didn't fit our uses, so we made our own. Another Issue related to this was "publishing to the bot's knowledge base," which is essentially having facty memorize the information it found, actually storing that information; however, this ended up being just a quick fix. The last problem related to indexing was extracting data from tables in documents, but we found a service offered by Azure to resolve this.

End Result

Throughout this semester, we all agree that working on this project has been the biggest highlight. Looking back at our process and now looking at the product we have created over these past few months, the experience has been very educational and rewarding. Along with mentors from SYNEX, we have created a very flexible chatbot whose domain of knowledge is not limited to its implementation, but instead to what documents you feed it. It is conversational by nature and versatile by design. This may have been our first experience with many of the technologies we molded facty with, but we worked together and created something to be proud of. Facty's abilities are not just limited to what we have laid out in this report, but every aspect can be further expanded on by whoever needs this bot.