Gindoff, Christopher, Davis, Banis, Kaur, Johney

Professor Wagenhoffer

CMPSC 487W 002

10 September 2023 - Requirements

14 October 2023 - Software Architecture

01 November 2023 - Testing & CI

1. Team Info & Policies

Project team:

Gabriella Gindoff: Backend Engineer
Adam Christopher: Frontend Engineer
Harkaren Kaur: UI Design/ Frontend Engineer
Treasure Davis: UI Design/ Frontend Engineer
Elizabeth Johney: Backend Engineer
Katherine Banis: Backend Engineer

Project Artifacts:

Git Repo: https://github.com/Gabi-Gindoff/PITA-repository
Proposal Doc: CMPSC 487W Project Proposal

Communication Channels:

Teams/Zoom to have weekly meetings to stay on track
Github to organize and store all project data and code
Messages group chat to communicate availability & updates

2. Product Description

Product Description:

PITA is a chatbot for the Penn State Abington IT Student Workers to utilize when they are faced with an issue they can not solve and need additional assistance with troubleshooting for common IT issues. This chatbot will enable the IT student workers to quickly retrieve answers to help faculty and students in an efficient manner. PITA answers common IT issues and questions, looks up details in the knowledge base, and provides step by step solutions. PITA serves as a fast and personalized virtual IT assistant.

4 Major Features:

1. Conversational interface between the chatbot and the user
2. Integration with a university database for responses
3. Simple, user–friendly interface that can be used efficiently

4. Fallback mechanism to provide relevant resources if the chatbot cannot find a direct answer to a question

2 Stretch Goals:

1. Expand the chatbot to other Penn State campuses
2. Add a login page to expand the program to all students in order to keep the IT departments data secure and only accessible to IT employees

3. Use Cases

Gabriella:

- Use case: serve as a resource for IT student workers when presented with questions beyond their technical knowledge
    - Actors: PSU Abington IT student worker
    - Triggers: Request from student or faculty for assistance
    - Preconditions: Chatbot with IT related information and a student who has a question
    - Postconditions (success scenario): Student worker receives an answer from chatbot and uses it to assist faculty or student
    - List of steps (success scenario)
        1. Faculty or student asks a question that IT student worker doesn't know how to answer
        2. Student opens up application to enter a query
        3. Chatbot searches database to find an appropriate response
        4. Chatbot returns the response to the user after validating
        5. IT student worker can use info to assist others
        6. IT student closes chatbot
    - Extensions/ variations of the success scenario: Validating that user question is within scope of chatbot knowledge, student following up on a question or clarifying chatbot's answer, validating chatbot answer
    - Exceptions/ failure conditions and scenarios: Student looks up a question that is not related to the PSU AB IT department

Adam:

- Use case: for IT admins to determine if more training is necessary
    - Actors: PSU Abington IT admins
    - Triggers: Admin notices that student is unable to adequately assist the requestor
    - Preconditions: Student was unable to assist with several requests over a period of time
    - Postconditions (success scenario): Admin views how many times IT student worker failed to assist before asking chatbot
    - List of steps (success scenario)
        1. Admin wants to determine whether further training is needed for student
        2. Admin opens the chatbot and searches for individual student workers to view number of queries submitted
        3. Chatbot searches database and retrieves relevant information

4. Chatbot returns information to Admin
5. Admin determines if more training is necessary
6. Admin closes chatbot

- Extensions/ variations of the success scenario: Verify that the user is an IT admin before allowing them to view the data
- Exceptions/ failure conditions and scenarios: An IT admin tries to get information on an employee who no longer works for the IT department, a student worker tries to view data regarding other employees

Treasure:

- Use Case: Removing outdated data from the database
  - Actors: PSU Abington IT Admins
  - Triggers: chatbot returns outdated information
  - Preconditions: Database with old information
  - Postconditions (success scenario): Only up to date information in the database
  - List of steps (success scenario)
    1. IT admin opens up chatbot and receives an outdated answer or is notified by student worker that chatbot returned information that is no longer accurate
    2. IT admin tells chatbot to remove a given piece of information
    3. Chatbot verifies that user is an IT admin
    4. After verification, chatbot either removes the requested piece of information or informs user that they don't have permission to remove data from database
    5. IT admin checks that outdated information is no longer in the database
    6. IT admin closes the chatbot
  - Extensions/ variations of the success scenario: User authentication to ensure user status is IT admin
  - Exceptions/ failure conditions and scenarios: student worker tries to delete data from the database

Katherine:

- Use Case: Add new information to the database
  - Actors: PSU Abington IT Admins
  - Triggers: IT department receives new software or has new information on old systems they want to add to database
  - Preconditions: Database with information and IT admin that has new information that is not in the database
  - Postconditions (success scenario): New information is added to the database and chatbot can now use that information in its responses
  - List of steps (success scenario)
    1. IT department receives new software/ system or piece of information
    2. IT admin opens up chatbot
    3. IT admin tells chatbot to add a certain piece of information
    4. Chatbot verifies that user is an IT admin

5. After verification, chatbot either adds the requested piece of information or informs user that they don't have permission to add data to the database
6. IT admin checks that the new information is found in the database
7. IT admin closes the chatbot

- Extensions/ variations of the success scenario: User authentication to ensure user status is IT admin
- Exceptions/ failure conditions and scenarios: IT student worker tries to add data to the database

Harkaren:

- Use Case: for IT department to keep track of common technical issues
  - Actors: IT faculty at PSU Abington
  - Triggers: IT admin want to know which rooms/ professors have the most tech issues, or want to know about recurring issues
  - Preconditions: IT department needed to fix room equipment or help professor with tech issue, database that stores with information
  - Postconditions/ success scenario: IT admins can view information about which rooms have faulty equipment or which faculty requests the most help
  - List of steps/ success scenario:
    1. IT Department wants to know what rooms or professors need assistance most often
    2. IT admin opens up chatbot and asks it for information regarding a particular piece of equipment, room, or staff member
    3. Chatbot responds with appropriate data that answers the query
    4. IT admin can use that information to decide if new equipment is needed
    5. IT admin closes the chatbot if nothing more is needed
  - Extensions/ variations of the success scenario: IT department finds that a room has faulty speaker and orders new hardware
  - Exceptions: failure conditions and scenarios: problem isn't documented correctly so admin doesn't have enough information to decide if new equipment is needed

Elizabeth:

- Use case: Authenticating user IT admin
  - Actors: IT Admin
  - Triggers: The user have to add or remove the data from the database
  - Preconditions: The chatBot should ask for valid credentials.
  - Postconditions (success scenario): The IT admin should have a valid credential that connects to the admin portal.
  - List of steps (success scenario)
    1. Open up the chatBot
    2. Ask the chatBot to edit the database
    3. ChatBot ask for IT credentials
    4. Check to see if the credentials are valid
    5. Verify/deny access to the user

- Extensions/ variations of the success scenario: If the user puts an unrecognized credential the chatBot prompts to retry. If the user exceeds the maximum login attempts, lock the account for a specific duration or require an account unlock process depending on security policies.
- Exceptions: If the user doesn't provide any credentials or provides unrecognized credentials.

4. Non-Functional Requirements

3 Non-Functional Requirements:

1. Usability - intuitive conversational interface where user can easily interact with the chatbot
2. Scalability - ability to handle multiple questions in a single queries or back to back questions
3. Security - ensuring there is no confidential information/data in the database that people who shouldn't have access to can get ahold of

5. External Requirements

1. Our chatbot will handle errors such as invalid user input by having a fallback method where we can provide them with other resources, ask for clarification, or inform them that PITA can't answer the question with the information it currently has.
2. Our product will be a web-based application that will have a public URL that others can use to access it
3. Our product will be buildable by others. We will provide step by step instructions and documentation for how to use our product.
4. The scope of our project will align with the number of team members assigned to developing our product.

6. Team Process Description

3 Major Risks:

1. Knowledge base content limitations - Risk that we cannot populate a robust enough IT troubleshooting knowledge base for the chatbot to provide useful responses. Quality of responses depends on comprehensive content.
2. Conversational AI limitations - Risk that the chatbot has poor natural language capabilities and cannot understand or converse fluently with users. This would cripple user experience.
3. Project scope creep - Risk that we keep adding features and capabilities beyond the timeline and resources of our team. Need to prioritize and limit features.

Risk Assessment:

1. The chatbot not being able to grab data from the database
    - ➢ Likelihood: Low
    - ➢ Impact: High
    - ➢ Evidence: We do not have it working yet, but assuming everything works as intended the risk of the risk occurring would be low with the impact being very high as its the base of the project
    - ➢ Steps to reduce likelihood: By doing significant testing, the issue can be limited/fixed
    - ➢ Plan to detect problem: If the chatbot does not give a proper response/the wrong response
    - ➢ Plan to mitigate if it occurs: Take chatbot offline and check code to ensure the connection between the bot and database is working properly
    - ➢ Changes: This wasn't a risk that was identified in our previous report but falls under the same idea of the chatbot limitations
2. The web scraper not being able to to grab relevant information
    - ➢ Likelihood: Low
    - ➢ Impact: High
    - ➢ Evidence: As of now the web scraper takes very long to run and grabs very little information. With more efficient writing, the scraper can grab the information it needs quickly.
    - ➢ Steps to reduce likelihood: Double check code to ensure the scraper is grabbing the correct information
    - ➢ Plan to detect problem: If the scraper gives improper data, we know an error occurred
    - ➢ Plan to mitigate if it occurs: If it occurs, we plan to look into the code for the scraper and make it more efficient / ensure it is scraping the correct resources
    - ➢ Changes: This wasn't one of the risks stated in the previous report however a part of this risk is having enough information on the teams channel that can be stored in the database
3. Database is able to be publicly accessed
    - ➢ Likelihood: Medium
    - ➢ Impact: High
    - ➢ Evidence: As of now, there are no security measures put in place to prevent people from going into the database. Implementing a login system is one of the priorities for the database
    - ➢ Steps to reduce likelihood: Make a login system to prevent unauthorized users
    - ➢ Plan to detect problem: The program detects logins and will log when a unauthorized person logs into the system
    - ➢ Plan to mitigate if it occurs: If credentials are incorrect, the program will not let the user into the program
    - ➢ Changes: Newly identified risk that falls under our security risk
4. Scope of project becoming too large
    - ➢ Likelihood: Medium
    - ➢ Impact: Medium
    - ➢ Evidence: With more and more ideas coming in, the scope of the project is growing to be very large and we do not have time to complete it all. We have decided to prioritize some objectives over others.

- ➢ Steps to reduce likelihood: Focus on the big picture and work on main features first
- ➢ Plan to detect problem: If we don't have a working chatbot with a connected database with some data, we know that the scope has become too large
- ➢ Plan to mitigate if it occurs: Take a step back and prioritize to ensure product has the main features developed by the end of the semester
- ➢ Changes: This risk has not changed since the beginning of the project but as we work on it we have more ideas so more potential for it to get out of hand

5. Chatbot grabbing irrelevant information for a given input
   - ➢ Likelihood: Medium
   - ➢ Impact: High
   - ➢ Evidence: With the chatbot grabbing information from the database, there is a good chance that it will grab irrelevant information for a user's input. I have been looking into the chatbot's code to see just how it grabs data to prevent this issue.
   - ➢ Steps to reduce likelihood: Perform various tests to ensure correct answers are given and use fallback method in case no data is found for topic being asked about
   - ➢ Plan to detect problem: Test it with multiple queries and if irrelevant information is returned we know there is a problem
   - ➢ Plan to mitigate if it occurs: Debug and add necessary info to the database or rework fallback method to catch it
   - ➢ Changes: This risk falls under the database limitations risk where the quality of the responses depends on the data being stored in the database as well as the chatbot's ability to use the data to return a correct response

<u>Software Toolset & Justification:</u>

- ➢ TypeScript, HTML, CSS for the frontend: Typescript adds type safety and scalability and is simple to learn, HTML and CSS provide webpage structure and styling. These languages work well together to create a clean webpage for the front-end development.
- ➢ MySql for the database: MySQL is open source and provides a simple yet robust data storage and querying for the knowledge base.
- ➢ Llama to create the chatbot: Conversational AI framework tailored for chatbots that provides easy to use natural language processing techniques and fully supports building chatbot conversations.
- ➢ GitHub for version control: GitHub enables collaboration and backup for code and supports team development with creating branches and pull requests.
- ➢ Visual Studio Code for the IDE: Free IDE, good for debugging code and supports Typescript.
- ➢ Jest for testing: Popular unit testing framework for Typescript and ensures code quality.

<u>Team Roles & Justification:</u>

- ➢ Gabi - Backend Engineer
  - This role is needed in order to create a database with the IT department's information. Gabi is suitable for this role as she has access to the information using the TEAMs chat as she currently works for the IT Department.

- ➢ Adam - Frontend engineer
    - This role needs to be filled to make a simple, easy-to-use webpage as well as integrating the chatbot. Adam is suited for this role as he has experience with creating web applications.
- ➢ Treasure - UI Design/Frontend Engineer
    - This role needs to be filled in order to efficiently design and create a visually appealing, user friendly interface. Treasure is suited for this role because she has experience designing and building web pages.
- ➢ Karen - UI Design
    - The UI Design role is for assisting with the development of a webpage and integrating the chatbot into the webpage. Karen is suited for this role because she has experience working with webpages.
- ➢ Katherine - Backend Engineer
    - This role is responsible for creating the database and inserting data that the chatbot needs to respond to user queries. Katherine is suited for this role as she worked with databases before and can use her knowledge from past experiences to help with the backend.
- ➢ Elizabeth - Backend Engineer
    - Backend is responsible for core functionality and infrastructure of the database. Elizabeth is good for this role because she is in a database design class so she can use those skills in this project and which give her more practice.

Team Structure:

Our team structure is divided into two subteams, front-end and back-end engineers: The team members working on the front-end are Adam Christopher, Hakaran Kaur, and Treasure Davis. The front-end team's responsibility focuses on designing and creating the graphical user interface that users will interact with. This includes the chatbot's appearance, layout, and user experience. Frontend Engineers focus on enhancing the overall user experience by ensuring the chatbot is intuitive, responsive, and visually appealing. Adam has been working on getting the chatbot up and running using Llama while Harkaran and Treasure work on developing and designing a website to host it.

The team members working on the back-end are Gabriella Gindoff, Elizabeth Johney, Katherine Banis. The backend engineers focus on gathering, organizing, and managing data for the Llama 2 chatbot. They serve as the backbone of the chatbot's functionality. They are responsible for integrating the chatbot with a SQL database, where they collect, store, and retrieve data relevant to the chatbot's operations. Gabriella has been working on developing a web scraping tool that can extract the data from the IT department's Microsoft teams channel while Katherine works on creating tables in the database using SQL. Both frontend and backend teams collaborate to create a cohesive and fully functional chatbot system by continuously communicating, exchanging ideas, and sharing insights to deliver a chatbot that meets user expectations and effectively retrieves and presents data.

Schedule/ Milestones:

1. Frontend/ UI:
    a. Develop a working webpage (9/25)
    b. Integrate the chatbot into the webpage using Llama (10/11)
    c. Style the webpage for the UI (10/25)
    d. Ensure the page is user friendly and easily accessible (11/15)
2. Backend:
    a. Develop a database (9/25)
    b. Add information from teams channel into database (10/11)
    c. Connect backend to frontend (10/25)
    d. Add ability to store information related to user and their queries (11/15)

Milestones, Tasks & Dependencies:

- Project Initiation (External)
    - Define project scope, objectives, and assemble the project team (Week 1)
    - Dependencies: None
- Frontend/ UI Development (Internal):
    - Create a working webpage (Week 3-5)
    - Integrate the chatbot into the webpage using Llama (Week 5)
    - Style the web page for user interface (Week 4-5)
    - Ensure the page is user-friendly and accessible (Week 4-5)
    - Dependencies: Project initiation must be completed and project must be approved before starting on the development process
- Backend Development (Internal):
    - Develop a database (Week 2)
    - Add information from Teams channel into the database (Week 5-6)
    - Connect the backend to the frontend (Week 6-7)
    - Add the ability to store information related to users and their queries (Week 8)
    - Dependencies: We can create a database and add data to it but the frontend must be completed before we can start connecting the backend to it since it needs a working webpage for integration. Database can be worked on in parallel with chatbot and frontend up to a certain point.
- Chatbot Development (Internal):
    - Integrate Llama chatbot framework (Week 2-3)
    - Implement the fallback mechanism (Week 6)
    - Enable user authentication for IT admins (Week 7)
    - Dependencies: To begin the only dependencies are the installations for running the chatbot, once it is fully working, it needs the frontend to host it on a website and it needs access to the backend/ database in order to answer user queries.
- Web Scraper Development (Internal):
    - Develop Python web scraper for Teams messages (Week 3-5)
    - Integrate Selenium and ChromeDriver (Week 4)
    - Test message extraction and integration with the database (Week 6)

    - Dependencies: Building the web scraper can run in parallel to frontend and chatbot development but once it can extract data from the teams channel, it needs access to the database to store the information.
- Testing and Integration (Internal):
    - Test chatbot responses and database interactions (Week 7-9)
    - Verify chatbot-user interaction in the web interface (Week 7-9)
    - Dependencies: Requires that frontend, backend, and chatbot development have reached a certain level of completion to ensure proper testing.
- Use Case Testing (Internal):
    - Run a full use case with chatbot interaction (Week 10-11)
    - Dependencies: Depends on the successful completion of testing and integration for a use case to be tested.
- User Training and Documentation (Internal)
    - Develop training materials and user documentation for IT department staff (Week 10-11)
    - Dependency: In order to create proper documentation, use case testing should be complete in addition to all other development needed to test the product.
- External Feedback Implementation (External):
    - Gather external feedback (Week 12)
    - Iterate and make necessary adjustments (Ongoing)
    - Dependencies: Depends on the completion of all testing in order to gather feedback and make necessary adjustments.

<u>Test Plan: Test-automation & CI</u>

- Test-automation infrastructure: Pytest
- Justification for test-automation infrastructure
    - Pytest is a widely-used and highly extensible testing framework for Python. It is known for its simplicity and powerful features. It provides a straightforward way to write test cases and has a rich ecosystem of plugins, making it a suitable choice for automating tests in Python projects. Its readable and expressive syntax facilitates effective test case design and reporting. Pytest is also well documented.
- How to add a new test to the code base.
    1. Open a GitHub codespace within the repository
    2. Open a new terminal
    3. Install pytest if not already installed (pip install pytest)
    4. Create a new Python test file by adding a new file (test_file.py)
    5. In the test file, write a function that contains your test cases.

```
def test_add_data_post(client):
    data = {
        'question': 'How do I use pytest?',
        'response': 'You can use pytest for testing Python applications.',
```

```
    'responseDate': '2023-10-31',
    'category': 'Testing'
  }
  response = client.post('/add_data', data=data, follow_redirects=True)
  # Check if the status code is a successful status code (e.g., 200 or 302 for redirection)
  assert response.status_code in (200, 302)
```

6. Run pytest on the test file in the terminal (pytest test_file.py)

- CI service and how our repository is linked to it
  - GitHub Actions is integrated directly into our GitHub repository. It is configured through YAML files (github/workflows/mainCI.yml) that define workflows, triggers, and jobs. Using github actions, we can specify when and how our CI pipeline runs based on events like pushes, pull requests, and more. In our case, the CI service will access our GitHub repository, and the triggers will be configured to execute the CI build when specific events occur, such as pushes or pull requests.
- Justification for chosen CI service.
  - GitHub Actions is chosen because our code is hosted on GitHub. It provides seamless integration, and it's a well-documented and widely adopted CI/CD service that supports a variety of workflows and customization options. Using GitHub Actions helps maintain all project-related processes in one place.
- A pros/cons matrix for two CI services that you considered.

	Pros	Cons
GitHub Actions	Tight integration with GitHub repositories simplifies the setup and management of CI/CD workflows, we chose it because our code is being hosted in GitHub, offers free usage for public repositories	Limited free minutes so we have to keep this in mind when running tests, free tier comes with restrictions which can impact the speed of pipelines, maximum execution time limit for workflows
Travis CI	Quick setup with minimal configuration, supports wide range of languages, high degree of customization with yml files to customize build processes	Limited amount of build time so can be pricey, does not offer the same level of integration and features as GitHub Actions when used within the GitHub ecosystem

- Tests Executed in CI Build:
  - As of right now only unit tests are executed in a CI build
  - We hope to add other tests, linting and code coverage checks
- Which development actions trigger a CI build.
  - Pushes to the repository
  - Pull requests (PRs) being opened

Documentation Plan:

- ➔ End-User Guide: This guide will provide detailed instructions for end-users, particularly IT student workers and other members of the Penn State Abington community. It will cover how to interact with the chatbot, use the web interface, ask questions, and troubleshoot common IT issues.
- ➔ IT Admin Guide: This guide will be tailored for IT administrators who are responsible for managing the system. It will cover tasks like database maintenance, user authentication, adding new information to the knowledge base, and handling data security.
- ➔ Llama Chatbot Integration Guide for Developer: As the chatbot component uses the Llama framework, this guide will detail the integration process, configuration, and customizations that developers may need to undertake.
- ➔ Web Scraper Implementation Guide for Developer: This guide will provide insights into the Python web scraper used to extract messages from Microsoft Teams. It will include details on setup, customization, and operation.
- ➔ Help Menu: Within the web interface, help menus or tooltips can be incorporated to guide users on using various features, chatbot commands, or UI elements.
- ➔ Wiki: A wiki or online knowledge base can be set up to provide extensive, searchable documentation. This could include FAQs, troubleshooting tips, and step-by-step guides.
- ➔ Error Messages and Troubleshooting Guide: A guide detailing common error messages and their resolutions can assist users and administrators in diagnosing and fixing issues.

External Feedback:

External feedback would be most useful when the webpage is accessible with the chatbot being able to access the database with information and towards the end of the project to ensure everything is working as intended.

7. Software Architecture

Overview:

The PITA project utilizes a client-server architecture model to establish a clear division of roles and responsibilities within the system. The server component handles data management, including the knowledge base, core chatbot logic, and user authentication. The client component represents the user interface, facilitating interactions with the chatbot. This architecture streamlines data processing, scalability, and maintenance while ensuring a user-friendly experience.

Components & Functionality:

- The chatbot component serves as the core of the system. It is responsible for receiving user queries and providing responses. The chatbot interacts with the database to retrieve information for answering queries and offers a fallback mechanism if it can't find a direct answer. It will also handle user authentication for IT admins.
- The frontend component is responsible for creating a user-friendly web interface where users, including IT student workers and IT admins, can interact with the chatbot. It provides the means

for users to input queries, view responses, and interact with the system. This component also integrates the chatbot into the webpage using the Llama framework.

- The backend component focuses on managing the system's database. It creates and maintains the knowledge base used by the chatbot. It stores information related to common IT issues and user queries. This component also connects the frontend and the chatbot, ensuring data is passed between them.
- The Web Scraper component is responsible for extracting messages from a Microsoft Teams channel. It authenticates, retrieves messages, and then passes it to the database to be stored and accessed.

Interfaces:

- The chatbot interacts with the database to retrieve information for answering user queries. It sends SQL queries to the database and receives responses containing relevant data, such as responses to common IT issues or user interaction history. This interface enables the chatbot to access and provide information stored in the database.
- The chatbot communicates with the frontend component to provide responses to user queries. It sends response messages to be displayed in the user interface and receives user queries and authentication requests. This interface ensures that the chatbot's responses are seamlessly integrated into the user-friendly web interface.
- The frontend component forwards user queries and authentication requests to the chatbot for processing. It sends user inputs to the chatbot, which analyzes and generates appropriate responses. This interface enables user interactions with the chatbot through the web interface.
- The backend component manages the database and communicates with it to store and retrieve data. It sends data insertion requests when adding new information to the knowledge base and retrieval requests to access stored information. This interface ensures the database is up to date with knowledge base data and user interactions.
- The web scraper component extracts messages from a Microsoft Teams channel and sends them to the database for storage. It communicates with the database to insert message data. This interface ensures that messages from the Teams channel are integrated into the system's database for processing and analysis.

Data:

Our database stores message data so all the messages that are extracted from the Microsoft Teams channel including questions and answers are stored in the database. It will include all information on common IT issues, responses, and knowledge base details. We hope to add user Interaction data, meaning all data related to user queries and chatbot responses. While the majority of the data that is being stored in the database is coming from the IT department’s teams channel, we will also add other information that is useful such as room quirks and other random facts or information pertaining to Abington’s IT Department.

Assumptions:

1. Consistency of Microsoft Teams Structure: The architecture assumes that the structure of Microsoft Teams, including the message format and data accessibility, remains relatively

consistent. Any significant changes to the Teams platform may require updates to the web scraper and potentially other components.

2. Structured Database Usage: The architecture assumes that data in the database, particularly the knowledge base, is well-structured and organized. It assumes that information is stored in a format that can be efficiently retrieved and processed.
3. Microsoft Teams as Primary Communication Channel: The web scraper is designed to extract messages from Teams, and the architecture assumes this as the primary source of data as it is the primary source of communication for IT student workers and IT admin.

Alternatives:

In our software architecture decisions, we first selected a relational database, such as MySQL, as our primary data storage solution. This choice is advantageous because relational databases are well-suited for structured data, particularly for our knowledge base that stores common IT issues and responses. They excel in handling complex queries, essential for searching and retrieving specific information, and provide robust data integrity enforcement. An alternative option, a NoSQL database like MongoDB, would have been more appropriate for unstructured data but less so for structured responses and might have limited querying capabilities.

For the chatbot component, we opted to use a dedicated chatbot framework, such as Llama, which provides pre-built components and advanced natural language processing (NLP) capabilities. This decision accelerates development, enhances our chatbot's conversational abilities, and benefits from community support. Alternatively, developing a chatbot from scratch would have been time-consuming, with extended development time and complex NLP implementation challenges. It also entails higher maintenance overhead as it might not match the evolving NLP field. Our choices align with our project's goals of efficiency and advanced NLP capabilities.

8. Software Design

Software Component Definitions:

- Chatbot Component:
    - The Chatbot Component consists of two primary parts: the Llama Framework and the Fallback Mechanism. The Llama Framework includes various packages and classes for natural language processing and conversation management. It manages user queries, processes them, and retrieves relevant information from the database. The Fallback Mechanism, on the other hand, provides alternative resources or responses when the chatbot cannot find a direct answer to a query. It ensures that users receive helpful information even in cases where the chatbot's knowledge is limited.
- Frontend Component:
    - The Frontend Component includes packages and classes for web development, such as HTML, CSS, TypeScript, and UI components. These elements collaborate to create a user-friendly web interface. The responsibilities encompass user input processing, message display, and interaction management. Additionally, the component integrates the chatbot into the webpage through the Llama framework, enabling seamless communication between the user and the chatbot within the web interface.

- Backend Component:
    - The Backend Component encompasses packages and classes related to database management and knowledge base creation. It includes database interaction classes, data models, and knowledge base management components. These parts are responsible for creating, maintaining, and organizing the knowledge base. They handle data insertion, retrieval, and ensure the database is up to date with message data and user interactions. The backend component acts as the bridge between the frontend, chatbot, and the database, facilitating data flow between these components.
- Web Scraper:
    - The Web Scraper Component utilizes Python, Selenium, and ChromeDriver for message extraction from the Microsoft Teams channel. It consists of Python scripts and web scraping classes responsible for authenticating, retrieving messages, and passing the data to the database for storage. This component is responsible for ensuring that messages from the Teams channel are efficiently extracted, making them available for chatbot responses and data analysis within the system.

9. Coding Guidelines

Coding Style Guideline:

Python https://peps.python.org/pep-0008/: The PEP 8 – Style Guide for Python Code is a great guideline for what we're doing. The contents on the left hand side make it easy to reference specific guidelines we may be looking for. Following the same guidelines will ensure consistency and allow for more clear and readable code that will help other members in the group follow along more quickly and easily. It will be easier to maintain and should reduce the chances of errors and bugs. A coding style guideline will also provide an objective set of criteria for which the code can be evaluated with. In order to enforce these guidelines, we plan on having team members perform manual code reviews before merging changes into the codebase.

SQL https://www.sqlstyle.guide/: After reading good things about Joe Celko's SQL Programming Style book, this site claims to be compatible with it. We liked the format, it's to the point, its simple, and it's very easy for us to reference to ensure our code is consistent. It gives examples and at the beginning, it has do and avoid sections which makes checking the general guidelines easy. Our plan to enforce these guidelines includes manual review and collaboration among team members. Code reviews, where team members check and provide feedback on each other's SQL code, can help ensure adherence to the SQLStyle Guide.