Final Report

JabberJack

Team Member:

Sara Harris, Tyler Zimmerman, Jiasheng Yang, Gabriel Proctor

Team Mentor: Felicity H. Escarzaga

Client: Dr. Andy Wang

Northern Arizona University

Version 1.0

Client Signature:

Team Signatures:

---

5 May, 2022

Contents

1.0 Introduction

JabberJack

Chatbots and other helpful bots are becoming increasingly more common throughout both the world of business and within academia [1]. Websites use chatbots to answer simple questions that visitors may have about products or services. There are even chatbots that are present on some Northern Arizona University (NAU) websites such as the one present on the Campus Health Services page [2]. Chatbots are not as complex as many people believe them to be; chatbots are not able to hold a conversation or answer extremely complex questions[1]. Even though chatbots cannot interact with humans in a human-like manner, there are a great deal of research teams working to improve the interactivity between chatbots and humans[1]. Chatbots can either be simple such as rule based chatbots which operate based on a number of rules, or they can be complex with artificial intelligence (AI) such as chatbots similar to Amazon's Alexa and Apple's Siri which learn from interactions to improve future interactions. The world runs on data, and the more data that a chatbot has available to it, the more helpful they become [4].

Chatbots are popular for customer consumption too, meaning that a great deal of chatbots can be found within regular people's homes. Amazon's Alexa is by far the most popular and the company holds about 62% of the market share within this industry [5]. In addition to Alexa, Google and Apple both have a comparable product. All of these chatbots are built utilizing complex computing called artificial intelligence. While artificial intelligence (AI) is the most popular type of chatbot, the more common chatbots are the ones that people can find located on websites. These chatbots are usually basic and only hold a small amount of information with little to no learning ability. They can answer basic questions about the website or about the company's service or product, so the company does not need to employ a real person for that role.

Our client Dr. Andy Wang is the dean of NAU's college of engineering, informatics, and applied sciences (CEIAS). Dr. Wang maintains and runs the programs contained within CEIAS and facilitates growth for both the college and programs in CEIAS. Dr. Wang is in the business of computing, engineering, and most importantly helping people. Dr. Wang deals with lots of people on a daily basis and is interested in helping them find answers to their questions efficiently.

2.0 Process Overview

JabberJack

The process overview will also introduce each team members' role, main responsibilities. In addition, this section will go through all tools and artifacts that were utilized during the lifecycle development.All of these components have given the team an edge in running smoothly and creating an intelligent chatbot which meets all functional requirements that Dr. Andy Wang expected.

2.1 Team Roles and Responsibilities

The team consisted of four team members. The team assigned the different coding jobs for each team member based on prior knowledge and known skill. Everyone keeped their primary roles and assisted some other team members who needed during the lifecycle development. The team structure stood as follows:

Member	Role	Coding Jobs
Sara Harris	Team Leader	Web Portal's Login and Register Page
Jiasheng Yang	Coder	Natural Language Processing and Web Portal's Management page
Tyler Zimmerman	Client Communicator	User Interface
Gabriel Proctor	Release Manager	Database

Table 2.1 Team members, roles and coding jobs

The team communicated with our mentor once a week at our mentor meetings. There was at least one coding meeting every week to combine the most recent modules together and arrange the next week's coding job, which aimed to make sure the system is working well together, and to fix bugs as soon as possible. The team also has a Discord server to communicate with other team members. For the team meetings, we also had some brief meetings via Discord or Zoom as needed.

2.2 Tools and Artifacts

Team JabberJack utilizes lots of tools as aid to development of the ChatterJack chatbot. GitHub is used for version control during the development; there is a public repository named

chatterjack to store codes. Every team member has access to the repository and is able to commit changes. For designing and drawing the flowcharts used in the document and presentation, the team used a software called draw.io. The team used Google Docs along with a team drive so that everyone can view and edit all of the documents and presentations. The core script was written in Python; Thus, PyCharm was the main IDE during the lifecycle development. Other team members who worked on the web portal used Vim as their IDE of choice.

The general life cycle of development was one the team would work together to plan out the deliverable (be it code or documentation), then each member would work on it whenever they had time, during team meetings the team would review and complete deliverables, then finally each member would review and approve the deliverables. When the deliverables were complete they would be sent off to the mentor and placed on our website.

We also utilized Discord as our main communication channel. Discord was also used as a meeting place for our team as some members were not always physically available. We held an in person meeting every Wednesday as well as met regularly over Discord to discuss, organize, and complete deliverables.

3.0 Requirements

JabberJack

This section illustrates the three requirements of the project: functional, non-functional, and environment requirements.

3.1 Functional Requirements

The ChatterJack is able to meet the following functional requirements that define its system capabilities. In order to gather these requirements the team had meetings with our client, Dr. Andy Wang, and held many discussions around what should be included in the requirements.

3.1.1 User Interfaces

The user interfaces are the portion of the chatbot where users, be it general users or back end users, can interact with the software itself. There are two parts that need to be addressed regarding this:

- User to chatbot question/answer interaction interface
- Administration/Management to database interaction interface

The graphical user interface has a text box for input. This is the most important function of the user interface as this is the only way for non-administrative users to interact with the chatbot. There is a submit button located to the right of the input prompt that is used to indicate that the chatbot can now search for the answer Once an answer has been found the chatbot then displays the answer to the user in an appropriate manner.

These functional requirements can be summarized as follows:

- Text box for user textual input
- Able to view the entire question at once
- Question Submit button
- “Pop up” for the answer
- Displays textual answers.

As for the web portal side there are several functions that were determined to be necessary for administrators. These requirements are as follows:

- Create new user accounts
- Page to input information
- View the system’s database of information:
    - Retrieve tables from the database
    - Display tables retrieved from database
- Add new entries to the database:
    - A Web Page to input new entries

- Remove data from the database:
    - View the tables from the database
    - Delete Entries
- Edit existing entries within the database:
    - View the tables from the database
    - Edit information contained within the tables

3.1.2 Authentication and Security

Authentication and security play an important role in any secure system; the ChatterJack chatbot is no exception. The authentication exists only on the server side of the application as there is no login required for "general" users of the chatbot. Administrators will need to be authenticated before they are able to access the chatbot's data. The data will be the **most** secure part of the chatbot as people, aside from admins, should not be able to edit the database.

- Only administrators can access the all entries within the database
- Includes viewing and editing
- Only authorized users are allowed to access the administrative portal.
- Authorized users that do not have administrator privileges can only access information pertaining to them

3.2 Non-Functional Requirements

Upon requirements acquisition it was determined that speed, security, and usability were of paramount importance. The chatbot abides by all the following guidelines.

3.2.1 Speed

The chatbot will take a maximum of 5 seconds; if an answer is not found within that time period the chatbot will respond with a default "I'm sorry I could not find the answer" statement. This is done in order to ensure that users are satisfied and do not become impatient.

3.2.2 Security

The ChatterJack chatbot will not allow unauthorized users to access any of the data contained within the database. An unauthorized user will just continue to get the same error and be unable to access the data. This will be done in order to protect both user data as well as the database.

3.2.3 Usability

The simplicity is mainly pertinent to the User Interface (UI). A simple UI will allow non tech savvy users to utilize it . The user interface should be so simple that anyone who can read and write will be able to use it to its full capacity. The chatbot should respond to questions in a perceivably polite tone; although the chatbot itself cannot be “polite” the language implemented is able to be written in such a tone.

3.3 Environment Requirements

There are no explicitly stated environmental requirements requested by the client, and as such it was determined that our team self impose environmental requirements, as this project sets a foundation for future chatbot projects at Northern Arizona University.

3.3.1 Operating System

Since there are no project restrictions put into place by the client there is only a single self-imposed requirement: This product must be able to run on Windows.

4.0 Architecture and Implementation

JabberJack

This section will demonstrate the architecture of the project and also the implementation of each model, including how each model works and how they work together.

The project was built with five major modules: user interface, the problem retrieval system, database, message generator, and web portal. The connections are shown in figure 4.1 below.

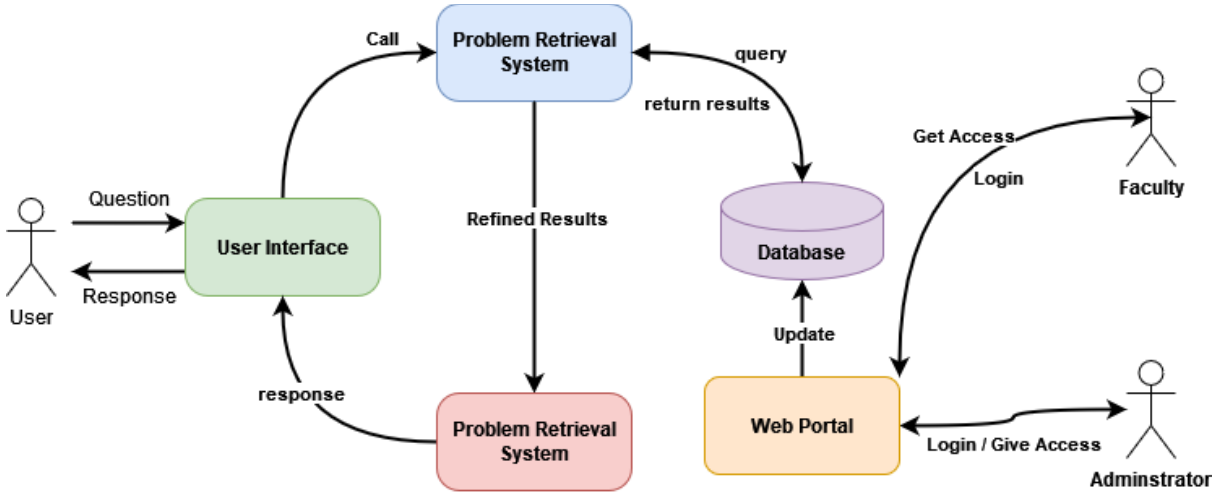

Figure 4.1 Architecture Overview

The user interface will receive a question from the user in the form of a string, then the problem retrieval system breaks it down into the question's basic elements. Then it queries the "online" database for the answer to the question. The database returns the results of the query back to the problem retrieval system, where it is packaged as a dictionary containing only the bare minimum information needed to get the message generator on the right path. The message generator then uses this dictionary to create a proper response. Finally the message generator then gives the response to the user interface which then relays it back to the user. The last component is the web portal which only has interactions with the database. On the web portal authorized users are able to update the information database.

The user interface is where general users (mainly students/staff) can interact with the chatbot side of the system. It receives input from the users and then receives responses from the message generator and displays them. Here users can type in their answers or there is a listen button they can press to input their questions in spoken form. Once it receives that response it then displays it and reads it out loud so that the user can listen. It has two connections; it hands information off to the problem retrieval system and receives information from the message generator.

The problem retrieval system is where the magic happens; it utilizes natural language processing or NLP to break down user input into its most basic elements. It breaks it down into a subject and questioning word or interrogative. It forms a connection to the database and

uses those basic elements to query it. It has connections to both the database and the user interface.

The database is where all of the information is stored. Ideally it lives on a server with the web portal but it can also be set up as a local database on a machine; which is how the final delivery was set up for the client. The database has three connections as shown in figure 4.1, it connects to the problem retrieval system, web portal, and the message generator. The database can be updated through the web portal and in general is just a place for all of this information to live.

The web portal can either live on a server or can exist on a local server. This is where faculty users and administrators can access and make updates to the database. The web portal only connects to the database and has no other function other than updating the database currently.

Finally the message generator is where responses are constructed. This is needed since in the database the information is just listed. It is not stored as human readable information or at least does not appear as sentences. The message generator uses the information returned from the database and some basic English elements to create a regular sentence that can be sent back to the user. The message generator has connections to both the database and the graphical user interface.

4.1 User Interface

The user interface is written entirely in python. It mainly uses wxPython as the main point of design but also utilizes other libraries which are listed and can be installed using the DwnLoadPackages.py file. Here is the link to the documentation on the library: https://wxpython.org/. Everything that the GUI needs is all in the GUI folder on the github repo; it consists of three main python files plus the DwnLoadPackages.py and a png file that contains the picture that is used to serve as the face of the chatbot. The first file is record_user_speech.py which is what is used to get input from the user by listening. It uses a library called soundservice to record the users questions. The next file is speech_to_text.py where the spoken question is converted into strings that the computer can actually read. Then finally the text_to_speech.py is where the response is spoken back to the users. Most of the code contained in these files are self documenting and there are comments that can explain what each is doing.

Essentially the only thing that the user interface is doing is displaying information, getting/converting information and then speaking information in a way that is usable to users. It “connects” with the problem retrieval system to pass off information that it has gathered from the user and gets information back from the message generator.

4.2 Problem Retrieval System

The problem retrieval system utilizes SpaCy to parse through the string given to it by the user interface module. Using a few methods native to SpaCy the software will pull out the words we want it to, in this case, the interrogative words, ”Who, what, when, where, how” as

well as words that could be considered names or classes. Using these bits of information, the problem retrieval system then queries the database looking for only the information it pulled, for example, if the question were “Who is Dr. Doerry?”, the problem retrieval system would categorize Dr. Doerry as being a name of a person rather than a class, then query the person database looking for only the column “who”. This information will eventually be passed off to the message generator for final construction into a human readable string.

4.3 Database

The database is the heart of the chatbot. This is where all of the information that the chatbot has access to lives. In the final delivery the database existed solely on a local machine database rather than an online server. While we would like for it to be deployed on a server this is not needed. There are two databases that exist in the project: the “phplogin” database and the “chatterjack” database. The phplogin database is only for user credentials and cannot be accessed by the chatbot.

The chatterjack database is much more complex. It consists of three tables each corresponding to a different “thing” the chatbot can have answers for. The person table is for people and contains eight columns: id: which is automatically incremented with each entry, person name, person spe name: which is like nicknames, where: office location, who, when: office hours, sex, and author. Then there's the class table which is for classes like CS480. In this table there are also eight columns which are similar to the person table: id, class name, class section, where, who, what, when, author. There is one more table, the organizations table or ORG for short it has 7 columns: id, org name, where, what, who, how, author.

The way the database is set up determines what questions the chatbot is able to answer. Meaning that since it is only able to have information about person: who, where, when, and sex it can’t really answer anything further like “Does Dr. Smith have a dog?”. This is possibly the biggest pitfall of the chatbot. There is a file called DB.sql which contains the structure and a bunch of information for the database that can be run to both create and populate the database. The database can be managed with basic SQL commands and can be changed but if changed everywhere that accesses it, meaning the web portal (every file) and the chatbot, will need to be changed since it’s heavily dependent on the structure.

4.4 Web Portal

The web portal is built entirely in: PHP, HTML, and CSS and that’s it. Originally the initial description of the project was just the chatbot, the team saw this as a problem since unless we implemented complete artificial intelligence the chatbot would be unable to “learn” new things. The web portal provided a solution, it is the way that the chatbot can “learn” new things. The reason that it was not built using some framework like Django was because of time constraints and the fact that the web portal was in general an afterthought. While the web portal probably is not as secure as it should be it has some general security that the team was able to implement.

Firstly there is a basic authentication system in place using the following files: auth.php, index.html(login page), register.php, register_user.php. The file, auth.php, is how

users are authenticated and how they login; it's connected to the login page. All that this file is doing is taking the username inputted and checking it against users in the database and then checks if the password matches the hashed version of the inputted password. To check the password it uses a built in function called password_verify. The other part of the authentication system is the register page; this is where administrators register new users. The form is the register.php page and the action is the register_user.php. It basically ensures first that there are no users with the inputted username and then hashes the password to store it.

The other part of the web portal is the actual database interface. There are different files for both the faculty and the administrators. Faculty files are named with "faculty" and administrator files just have general names. The file index.php is the home page for admin members. Each file function corresponds to their names.

In general the web portal is not well organized; there were ways to better organize the files but these actions were overlooked. There are two database files: accountsdb.php and db.php. The db.php file was well used where the accounts database file is not used but anywhere the accounts database is being accessed could be replaced by just including the accountsdb.php file.

Most files are well documented with comments describing what exactly the code is doing. There are some "extra" files that are not worked into the website: change_Pass.php, change_pass_action.php, activate.php, and test_hash.php. These are incomplete features and testing files.

4.5 Message Generator

The message generator is the module within the chatbot responsible for constructing a human readable string to be passed back to the chatbot user. While it works closely with the problem retrieval system, there is only one file within this module, message_generator.py. The chatbot creates this string by taking the dictionary created by the problem retrieval system, and depending on the interrogative stored within this dictionary, the message generator goes through a series of checks, to determine what type of answer should be returned. The message generator determines whether to reply with masculine pronouns or feminine ones depending on the values within the dictionary created by the problem retrieval system. While not implemented, there is a message generator variation file that attempts to expand upon these binary "male" or "female" base pronouns.

The message_generator_variation.py does the same thing as the message generator with the exception that it utilizes a "pronouns" column that was never able to supercede the "sex" column due to time contrains. The goal of this variation file was to give more variation to the way the chatbot replies, and was an attempt to be more inclusive, allowing faculty to input their pronouns instead of assuming gender from sex.

5.0 Testing

JabberJack

This section contains information about how the software was tested and how that impacted development. The general process of testing was testing the most important features and then moving onto other ones. The adopted strategy was a test as we build; so once a feature was finished being built the team would run tests on how it was working as well as testing how it works with the rest of the system.

5.1 Unit Testing

Unit testing was mainly focused on the natural language processing module and the user authentication system. The team felt like these two systems needed to be focused since without them neither the chatbot nor web portal will function as intended.

5.1.1 Natural Language Processing

The natural language processing system is the core of the ChatterJack chatbot and forms the chatbot's "understanding" of classes, people, organizations and interrogatives. This unit consists of two main classes; one is used for grabbing the interrogatives and subjectives, which is built by a python library named spaCy; another one is used for correcting the input and implementing fuzzy finding, which is built by a python library named sklearn and calculates the tf-idf. And all of them need to be tested to ensure that the chatbot is "understanding" input questions in a way that allows it to grab the correct answers needed for response.

By using unit testing, the team can ensure that the chatbot is extracting the correct information from user questions. Below table 5.1.1 shows all of the tests for grabbing intention class, inputs, expected outputs, and whether they passed or failed.

Function Tested	Expected	Outcome
`segmentSentence()`	Use token from spaCy segment the input sentence	**PASS**
`segmentList()`	Store each word in a list from token result	**PASS**
`interrogative()`	Grab the interrogative	**PASS**

`person()`	Grab the normal person using entity from spaCy	**PASS**
`nau()`	Grab the normal person using entity from spaCy	**PASS**
`expandPerson()`	Grab the specific name, eg: Dr. D	**PASS**
`classes()`	Grab the specific name using regular expression	**PASS**
`intention()`	Store all grabbing information into a dictionary and corresponding labels	**PASS**

Natural Language Processing Unit Tests Table 5.1.1

By using unit testing, the team can ensure that the chatbot is correcting the fuzzy input to the correct information stored in the database. Below table 5.1.2 shows all of the tests for correcting fuzzy input class, inputs, expected outputs, and whether they passed or failed.

Function Tested	Expected	Outcome
`tf()`	Calculate the tf-idf value of inputting and standard information	**PASS**
`compare()`	Store the inputting information and tf-idf value in a dictionary	**PASS**
`perCheck()`	By tf-idf value, judge whether correct the inputting information (person and organization)	**PASS**
`claCheck()`	By tf-idf value, judge whether correct the inputting information (class)	**PASS**

Natural Language Processing Unit Tests Table 5.1.2

5.1.2 User Authentication

User authentication exists so that the system is protected and so that the system can identify users. The management system for user authentication is written in PHP and utilizes the server's database to create and store user credentials. Users are required to request an account and from there are able to login and access certain elements of the database based on who they are. There are three major parts of the authentication system that need to be tested: registration, login, and logout. A manual test will be run to ensure that users are able to be registered, login to their accounts, and logout of their accounts.

To do this we will use a version of “black box testing”, a type of testing that tests functionality rather than code; the code is not visible hence the name blak box. The team will test the functionality of the authentication system and not directly test the code.

Registration Flow Test

- Administration will create a new account by filling out the registration form
    - Expected : “Account Creation Successful” prompt, **SUCCESS**
    - Otherwise: **FAIL**

Login Flow Test

- Login using the credentials just created
    - Expected : Homepage display, **SUCCESS**
    - Otherwise : **FAIL**

Logout Flow Test

- Click the logout button present on the homepage and check if session is successfully cleared by trying to access the URL of another page
    - Expected : Return to login page if session is properly cleared, **SUCCESS**
    - Otherwise: **FAIL**

System Tested	Expected	Outcome
Register	“Registration Successful”	**PASS**
Login	Homepage display	**PASS**
Logout	Login page display	**PASS**

Authentication Unit Tests Table 5.1.3

By utilizing these three manual tests the team can tell where something is failing. If it fails in registration then there is a bug in the creation of a new user entry in the database. If it fails in login then there is an issue drawing user information from the database. If it fails in logout then the session is not properly cleared and needs to be refactored. When all tests pass then the authentication system works as expected. Completing these tests are necessary to ensure that user information is correctly being stored and drawn from the database; it is done to ensure that user information is being protected and properly handled.

5.2 Integration Testing

In figure 5.2.1 we can see all of the connections that are made between the 5 separate modules. Connections that have passed their testing are shown here in green; all connections have passed in a local environment.

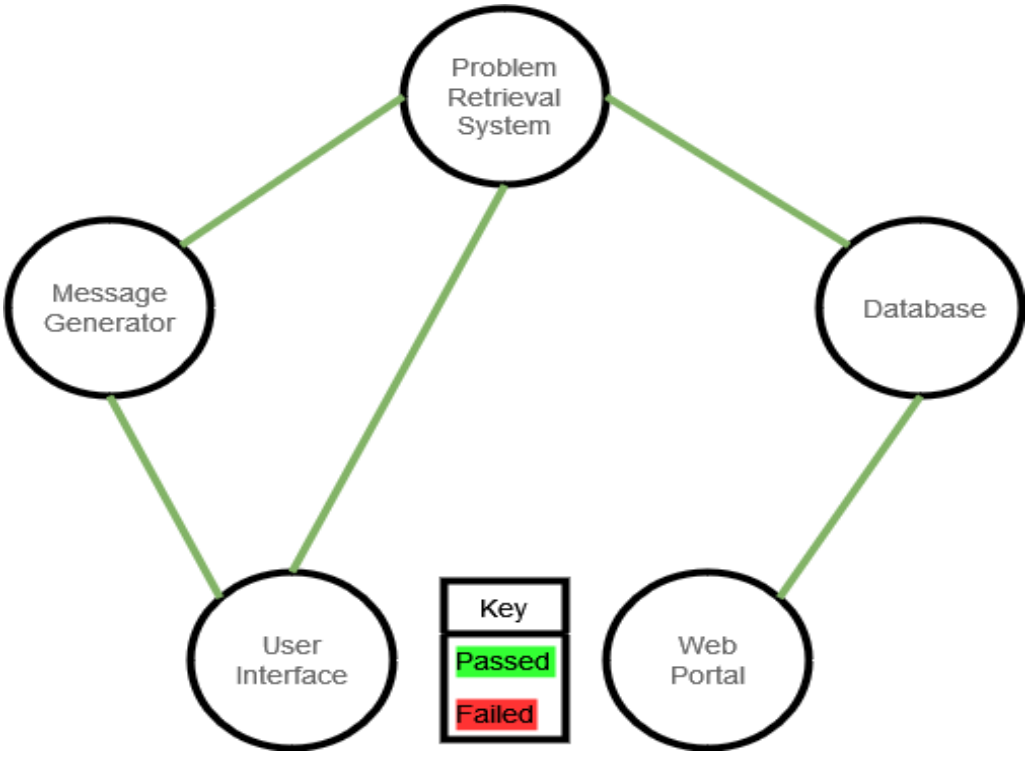

Figure 5.2.1 Integration

The software was passed onto the client as built in a local environment meaning that there was no online or cloud computing service utilized in the final delivery.

5.3 User Testing

The team will separate the end user into three different groups, and each group has five to six end users who will participate in different testing tasks; no further information about the product will be given to them as it needs to be intuitive. Users will then be asked to think aloud as they interact with the product. Then the team will take notes on how users are using the product, what they think of the product, and will gauge how much users are enjoying the product.

5.3.1 General User

The general user, including NAU faculty, NAU student, visitor, or anyone who wants to know some information about NAU, will only be able to interact with the chatbot. This user is able to ask questions either by typing out the question into the user interface or by speaking aloud.

Five to six general end users were asked to interact with the chatbot by asking questions; no further information about the product was given to them as it needs to be intuitive. Users were then asked to think aloud as they interact with the product. Then the team will take notes on how users are using the product, what they think of the product, and will gauge how much users are enjoying the product. Once the user is “done” working with the product they

will be asked to rate several characteristics on a scale of 1 to 10. They will be asked to rate: Easability, enjoyment, and engagement. The user interactions with product should following these functionalities:

- User can ask the chatbot a question via text/speech input
- Receive a correct answer shown on the screen and read aloud it
- The conversation should feel like the human interaction
- It can handle the multiple different inputs for the same question
- Return “Sorry, answer not found” message when it cannot find the answer to the user’s question

Results shown in table 5.3.1:

General User	Easability Rating	Enjoyment Rating	Engagement Rating
01	7	9	10
02	8	10	10
03	9	10	7
04	10	8	9
05	10	7	8

Table 5.3.1 General User Ratings

Some user quotes:

- “I think it’s really fun to use, I like the accent on the chatbot it’s funny” - User 04
- “The listening function is kind of annoying it didn’t hear me right once” -User 01
- “It was pretty clear on how to use it but for me it wasn’t really that fun” -User 05
- “I wish it had answers to more questions it seemed kinda limited” -User 02

This leg of testing got pretty decent reviews as shown in the table. Based on this testing we improved the message generator to give it better response structure. While we had some complaints that the range of questions was a bit limited, we really did not have any way to broaden that scope without scraping a large section of the project.

5.3.2 Faculty User

The faculty for this application can query or modify their own personal and course information. He/She cannot see or modify other faculty's information. They must get permission to get access to an account. When they login into the administration system it will link to the faculty manage page where they can manipulate data they have access to in the database.

Five to six faculty members were asked to manage their own information. All selected faculty members should start the testing from registering. This testing should pay more attention to the simplicity of operation and design of the web portal. They were asked a scale of 1 to 10 for both characteristics. The user interactions with product should following these functionalities:

- Faculty can request an account be made with given credentials
- Check their own register information, including the account type and username
- Login to the management system
- Insert and modify the information according to the selected table, PERSON or CLASS
- For personal information, they can modify their job, office room, office hour, and specific name
- For course information, they can modify class time, location, definition, and professor name
- Logout of their account

Results shown in table 5.3.2:

Faculty User	Operation Simplicity Rating	Design
01	6	5
02	7	6
03	9	5
04	9	3
05 (after changes)	10	7

Table 5.3.2 Faculty User Ratings

Some user quotes:

- “It was really easy to use I mean it seemed pretty self explanatory to me” - User 03
- “It’s pretty unusable I mean how do I even navigate this” - User 01
- “It looks pretty bad, it would look nicer if you added some color instead of having just white tables” - User 04
- “Wait so I can’t delete data?” - User 09
- “I really like the nav bar, the icons look really professional” - User 05

Based on user feedback we made some changes to the faculty page. First we got some heavy feedback on the fact that faculty users are unable to delete their own entries so we quickly added that functionality. We found that most of our feedback was based on design and how that can be improved; based on that we changed the look of our web portal by giving it some color and making the login page more appealing. We also got a ton of feedback on how the only way to navigate through the website was with back buttons and links so we added in a navigation bar that was accessible from the entire website which should greatly

improve navigational abilities. While many thought that it was pretty easy to use, the design needed work.

5.3.3 Administrator User

The administrator for this application has all permissions to manage all Question and Answer (QA) pairs from the database, including searching, deleting, inserting, and modifying. They also must get permission when they login into the administration system.

Five to six administrators will be asked to manage all QA pairs of three tables from the database. All selected administrators should start the testing from registering. This testing should pay more attention to the simplicity of operation and beauty of the web portal. They will be asked a scale of 1 to 10 for both characteristics. The user interactions with product should following these functionalities:

- Administrators need to request administrative access for an account
- Check their own register information, including the account type, password, and user name
- Login to the administration system
- Search, insert, modify, delete the information according to the selected table, PERSON, ORG, or CLASS.
- For search operation, if they do not type the specific searching information, all the information will be displayed
- For insert operation, there should be the information shown on the text box
- For personal information, they can modify their job, office room, office hour, and specific name
- For course information, they can modify class time, location, definition, and professor name
- For organization information, they can modify its name, location, director, and how to go there
- Logout of their account safely

Results shown in table 5.3.3

User	Operation Simplicity Rating	Design
01	5	3
02	6	5
03 (after changes)	9	8
04 (after changes)	10	9
01 (after changes)	8	7

Table 5.3.3 Admin User Ratings

Some user quotes:

- “Honestly this site looks awful it could use a lot of work on that end” - User 01
- “How do I go back…” “Oh I have to use the back button, thats dumb” - User 02
- “Oh you really dressed it up from before looks a lot nicer” - User 01 (after)
- “Wait, can I even change my password” -User 04
- “I mean it all makes sense to me” - User 03

Based on this feedback from the initial testing with 01 and 02 we needed to change the way that it looked. Before changes the website barely had any CSS and there was not a navigation bar. While we can only do so much we really tried to improve the design. Also based on this feedback we added a change password button, we had not even thought that it was needed until user 04 had said something about it. As was similar with the faculty users, user 02 found it difficult to navigate the site since there was no navigation bar so that was added as well.

6.0 Project Timeline

JabberJack

This section will go through the major milestones during the lifetime of this project. In Fall 2021, the team focused on the requirements and determining feasibility and came up with the basic framework for the project. The 2022 Spring semester was mainly about software design and development. Details are below:

6.1 Fall 2021 Semester

There are four main milestones in the project for the Fall semester, including: Technological Feasibility Analysis, Requirements Document, Design Review 1, and Technological Demo. All of these set the foundation for the next semester to generate a chabot system. Here are some explanations for them:

- **Technological Feasibility Analysis**:
  List all technologies that the chatbot will be used for, explaining why the team chose them.
- **Requirements Document**:
  List all requirements including functional, non-functional, and usability requirements and the architecture about how to achieve them.
- **Design Review 1**:
  Present initial problem and general solution. In addition, introduce all the requirements and techs the team chose.
- **Technological Demo**:
  Test and develop all the technologies that the team chose and how far the team reached.

6.2 Spring 2022 Semester

Figure 6.2 focuses on the development plan this semester so that JabberJack can finish the chatbot system and web portal to meet all the functional requirements that Dr. Andy Wang expected.

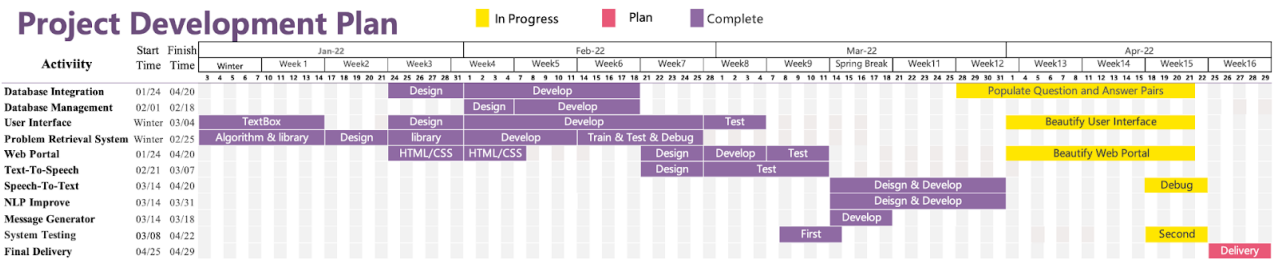

Figure 6.2 Project Development Plan

Here are explanations of the project development plan:

- **Database Integration (01/24/2022 - 04/20/2022)**
  As the database is the first step to implementing the key requirements, JabberJack began to collect and integrate the data set of Question and Answer (QA) from the end of 2022 January. This was finished by February 18th. JabberJack utilized the QA pairs as the corpus to train the chatbot so that it could be the intelligent brain.

- **Database Management (02/01/2022 - 02/18/2022)**
  The database was made up of two parts: local and online databases. The database management system tested with the administration system, which could update the Q/A through the administration system and online database.

- **Problem Retrieval System (Winter - 02/25/2022)**
  The Problem Retrieval System is the core of the chatbot, which affects the customer's satisfaction directly. It utilized the spaCy and saved corpus to train the model. JabberJack spent a lot of time making sure that the system could return the instantaneous and accurate answer to the Message Generator.

- **Web Portal (01/24/2022 - 04/20/2022)**
  The overarching component of the administration system is the web application, which consists of four parts: login page, administration page, register page, and faculty page. This process integrated testing with the database system management.

- **Text-To-Speech (02/21/2022 - 03/07/2022)**
  The team used the Python library named TTS to show the answer to the question both in the chabot window and voice.

- **Speech-To-Text (03/14/2022 - 04/20/2022)**
  It is very complicated to identify the speech to text. JabberJack utilized a lot of Python libraries to achieve the client's requirement.

- **User Interface (01/24/2022 - 04/21/2022)**
  JabberJack started designing the user interface (UI) from January 24th. The basic design should focus on the login window and chat window. The higher level should be a funny face shown on the screen. This process integrated the Message Generator.

- **Message Generator (02/07/2022 - 03/04/2022)**
  The Message Generator integrated with the Problem Retrieval System, User Interface, and Database. JabberJack utilized it to debug and test the system so that its answers are always reasonable and polite.

- **Improve NLP (03/14/2022 - 03/31/2022)**
  The team tried our best to improve the Natural Language Processing (NLP),which makes sure that the ChatterJack chatbot is as intelligent as it can.

- **System Test & Final Delivery**
  There are two system tests, one for examining the MVP, the other for the final product delivery.

7.0 Future Work

JabberJack

This project has the potential to grow into something much larger than we were able to build this year. In this section there are some suggestions and routes that we wanted to take the project.

7.1 Long Term Goals

These are some of the goals we had for the project that we felt were bigger than what we could accomplish within the time that we had this year.

7.1.1 Louie Robot Integration

The ChatterJack chatbot has immense potential for extending functionality that will integrate a physical mobile bot in the form of Louie the Lumberjack. The Louie Robot is a project that was developed in parallel with this one; their project website is: https://sites.google.com/nau.edu/nau-louiebot-capstone/defining-the-problem?authuser=0. Hopefully future teams will be able to successfully integrate the two projects. The two projects were both created with the intention of eventually being picked up by future teams and merged. Unfortunately the merge was unable to happen this semester. We feel like this would be a massive undertaking since it will take teams from multiple disciplines in order to integrate them properly.

7.1.2 Artificial Intelligence

Another aspiration that the team had for this project was artificial intelligence. While It would have been amazing to be able to implement it right off the bat, due to lack of experience and time constraints the team chose to focus on what they could accomplish. The ChatterJack has been designed with future work in mind; each section is modular and can be taken out and interchanged with future work. With artificial intelligence the ChatterJack will become easier to interact with and incidentally more enjoyable to do so. It will become able to answer a plethora of questions and not just the ones that were anticipated by the team. One lofty goal for this project that was expressed was to have something similar to Amazon's Alexa; while the capstone projects are college students the team feels that this project could come close to that goal.

7.1.3 Multithreading

The chatbot in its current state, while working, could be improved greatly through the use of multithreading. Right now the way that the chatbot works is that it is supposed to grab the answer from the database, then display it on the graphical user interface, and **then** speak

the answer aloud. But what is actually happening is the answer is starting to display, then gets interrupted by the chatbot speaking the answer aloud. We did not realize this to be a problem until after it was deemed too late to actually fix it. The idea that we had was to have things operate in parallel with each other so that the chatbot can do more than one thing at a time. Multithreading would solve a lot of the problems that we've encountered within the project. Firstly the program could "listen" on a separate thread which would solve the issue of the program becoming overwhelmed when trying to listen to a spoken question. Secondly it would allow the chatbot to both display and speak the response at the same time; which currently they overlap with each other. Parallel programming is a difficult task to complete in the final strides of the project but the team feels it can be implemented seamlessly into the project.

7.2 Starting Points

Here are some starting points of features and fixes that we wanted to accomplish with the project but either ran out of time or needed more polish before being completely usable.

7.2.1 Forgot Password Fix

The first major feature that needs work is the forgot password feature. The team did not think of this until after user testing was run, this gave the team only 1 - 2 weeks to complete the feature. The code is present in the source code within the forgotpass.php file. Since the team did not have a server for the final product they were trying to implement it so that it would work without one. This is tricky since the built-in mail function in PHP **needs** a mail server so without a server using it becomes difficult. PHPMailer is a package that can be used in PHP to use an outside or third party mailing service like outlook or Gmail. Within the source code there is the package labeled PHPMailer, inside is examples on how to accomplish using the package properly. There is also a test file in the web server that tests the randomly generated password as well as the mail function (which does not work). With the fix in the mail function it could become completely automated; from requesting access to actually gaining the credentials.

Once this becomes automated Admins on the web portal could click a single button and approve access on their dashboard. This would make the product more usable in the sense that not everything would be completely reliant on the involvement of a single admin or groups of admins.

7.2.2 Web Portal Quality of Life Updates

The web portal is not perfect and there needs to be some general updates in order for it to completely fit for everyday use. Firstly, the look of the actual operations of interacting with the database is a little non-intuitive which making everything intuitive was a massive deal for the team. In figure 7.2.2.1 there is an example of what administrators see when inserting new data into the information database. This is also the same as what a faculty user would see.

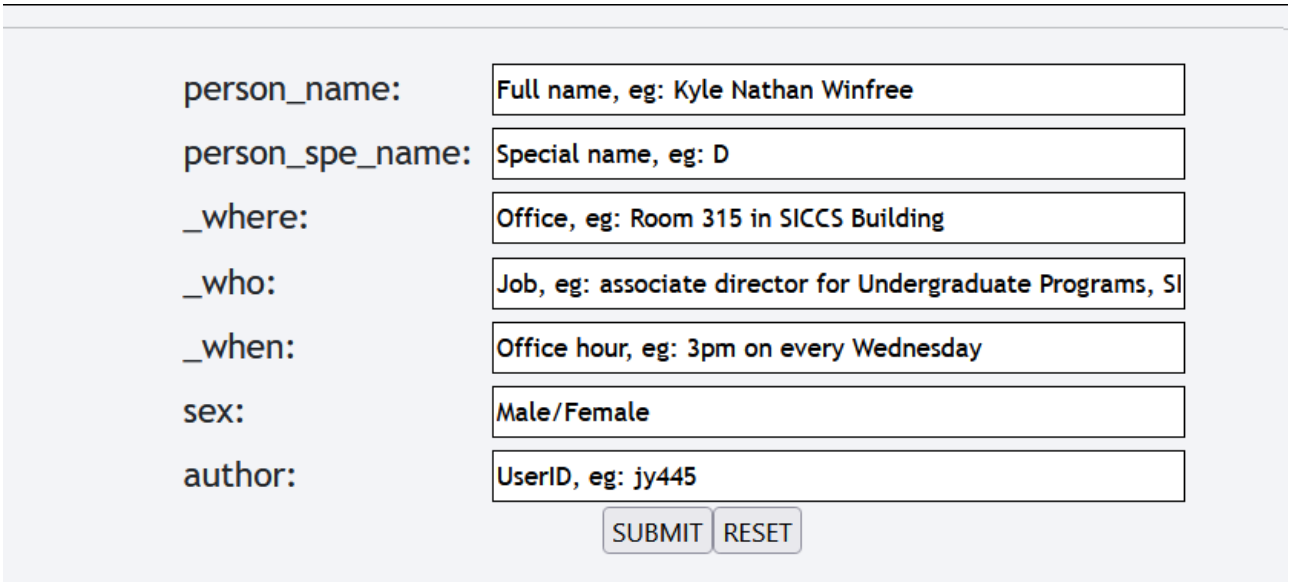

Figure 7.2.2.1 Admin Insert

Looking at this as a developer there does not seem to be a big issue but looking at it as a general faculty user it becomes evident that there is a problem; “What is ‘person_spe_name’?”. The titles of what is being inserted needs to be changed into something more user friendly, this also applies everywhere that these titles are being used.

Next issue is the inserting function. In figure 7.2.2.1 each text box is filled with information as examples of what could be in these boxes. How it works currently is that the database needs all of these fields to be something, meaning that it cannot be empty, but instead of substituting unfilled out information the team leaves the example information. So if a user was to just click submit without filling out anything it would be all of that example information in the database. The fix to this would be if a user did not fill out the field either force them to put something there or autofill something like **NA** or **TBD**.

Another massive problem is that when an administrator is deleting entries there is a dropdown menu that displays all information in the selected table, you can see this is in figure 7.2.2.2.

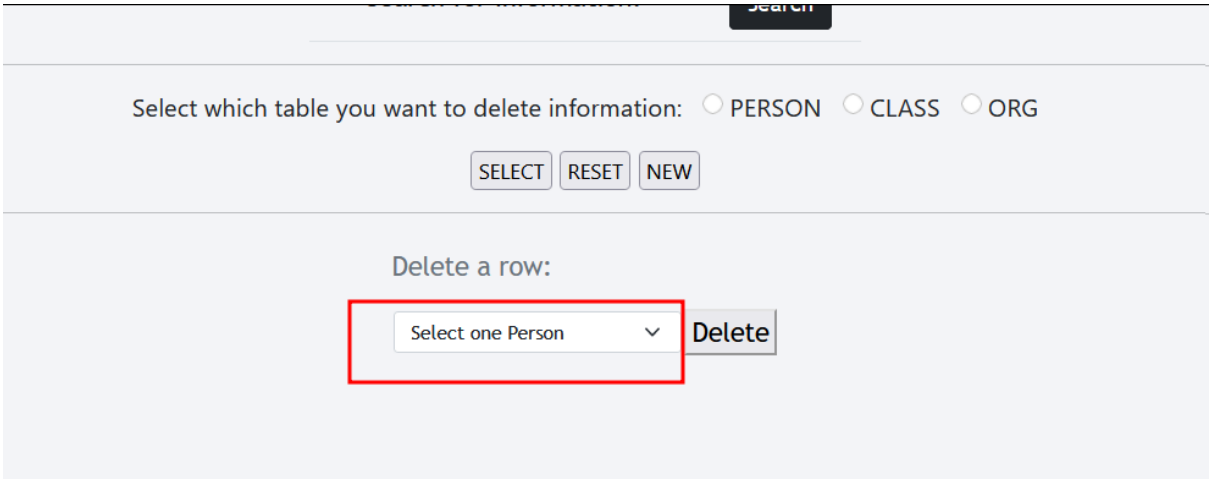

Figure 7.2.2.2 Delete Dropdown

This is annoying because an administrator will have to read through hundreds of entries to find the one that they want to delete. This is not an issue when an administrator wants to update an entry though since we implemented a search in that section. The update would give the delete function a search so that the dropdown menu would be reflective of that search.

7.2.3 Continuous Connection

Another problem that became apparent near the end of the project was that the chatbot did not have a continuous connection to the database. The way that it works currently is that when updates are made on the web portal the chatbot needs to be restarted in order to access that new data. Making a continuous connection with the server the chatbot does not have to be restarted every time for things to update. The idea is to make it so the chatbot can be working and receiving updates as it is answering questions.

7.3 Other Updates

There was very little direction on how the project was going to be built so the team developed a lot of ideas; while the main ones were implemented there were lots of features that the team could not implement. While this is not an exhaustive list it is some of the ideas that the team wanted to develop.

- Feedback system: the team wanted there to be some way to rate the answers to questions, this could allow faculty and administrators to detect when something was out of date
- Automated sign up: completely automate the signup process so that administrators can just approve accounts rather than set up accounts and manually have to enroll users
- Other languages: the chatbot is an english only chatbot, implementing other languages could allow it to reach a wider audience and allow more students to interact with it

- Implement some automatic basic web scraping: while there is a web scraper that our team developed it has to be run manually
- An online version: while it's fun to be able to walk up to a chatbot and interact with a chatbot it could be beneficial to deploy the chatbot online as well

8.0 Conclusion

JabberJack

Our client Dr. Andy Wang is the dean of the College of Engineering, informatics, and applied sciences which means part of his business is ensuring students, faculty and visitors have access to the information about the college. Right now at Northern Arizona University there is a disconnect between people and information. Information is difficult to access and within a world encompassed by the internet having quick access to information is a necessity. With chatbots like Amazon's Alexa and Apple's siri people have instant access to information at all times. Currently to get information people would need to:

1. Find someone to ask a question; this is not always possible since employees are not always present in buildings and they have their own tasks to complete
2. Search the internet; this is difficult since the NAU websites are difficult to navigate and sometimes the information present on them could be out of date
3. Email departments; email is the new snail mail and waiting for answers is sometimes not an option, these emails are sometimes even never answered or forgotten about

The ChatterJack chatbot can help bridge the gap between people and information.

- It is physically present in the engineering building on NAU campus meaning that it's always available to answer questions
- Provides a way to bring information to a single space instead of scattering information out on the internet with an information database
- Is able to respond to questions in a 5 second time frame

The ChatterJack is able to give instant answers to questions instantly eliminating the waiting period or work that currently goes into getting answers. The ChatterJack will allow the engineering college as a whole to function more efficiently. By utilizing natural language processing it is able to understand user questions and answer instantly.

The ChatterJack can become the start of physical chatbots here on Northern Arizona University campus. This project really sets the foundation for a much larger idea of academic chatbots on college campuses. The team feels that the project accomplishes all that it set out to fix and can continue to grow into something much bigger than what it is right now. The team worked well together to provide this software and is excited to one day see it in use here on NAU campus. This capstone course has allowed us to put everything learned in our computer science courses to use and create something brand new. We hope to see this project continue to grow and help students learn new things.

9.0 Glossary

JabberJack

Terms that were referenced in this document:

9.1 Appendix A: Development Environment & Toolchain

Here is a few important pieces of information for future development teams; it includes the minimum hardware requirements to how to set up the development environment.

9.1.1 Hardware

The software is only currently functional within a windows operating system, It is partially operational within Mac OS and how compatible it is within a Linux environment is unknown. While we do not think that there is anything hindering it from being semi functional in a Linux environment we have not tested it on Linux. The reason for this is some OS specific libraries used in Python, specifically the ones used for listening and responding. The Windows machines that it does work on all have an intel celeron or higher, which we think are the minimum requirements for the software.

9.1.2 Toolchain

The team mainly used PyCharm in developing the chatbot, no teammates used any plugins. Vim was also used but any editor can be used to develop. One major tool that was used in both the final product delivery and development was XAMPP. XAMPP provides a local apache server so that you can run PHP code and MySQL database on.

9.1.3 Setup

This section will walk you through how you can set up your machine in order to develop the product. While you can use any editor you want this walkthrough will show you how to set up PyCharm as the main Python editor.

Python

1. Download Python version 3.7.9 from this link https://www.python.org/downloads/release/python-379/ make sure that it's the 64 bit since spaCy requires that

PyCharm

PyCharm is a helpful editor that will set up a python environment as well as run your code.

1. Download the executable and download PyCharm https://www.jetbrains.com/pycharm/ you can use all of the default install options
2. Once it's complete you can use it to open python projects and make edits, it's also helpful since it helps with tabbing and catching errors before the code runs

3. Once you've downloaded all of the source code from either Github or from the thumb drive provided to Dr. Wang you can run the DwnLoadPackages.py file which will install all of the needed python libraries for the chatbot
4. run ChatterJack_GUI.py and that should run the chatbot GUI; you will need to have the MySQL and Apache turned on or else it may throw an error about the database, how to install Apache and MySQL is explained in the XAMPP section

XAMPP

XAMPP is a helpful tool that creates a local server where you can run and execute PHP code; this will allow you to test the web portal side of this project.

1. Go download XAMPP from here https://www.apachefriends.org/download.html
2. All of the default options work; although it may be helpful to put it somewhere that you can access easily like your desktop
3. Once it's installed you can run it from the control panel shown in figure 9.1.3.1

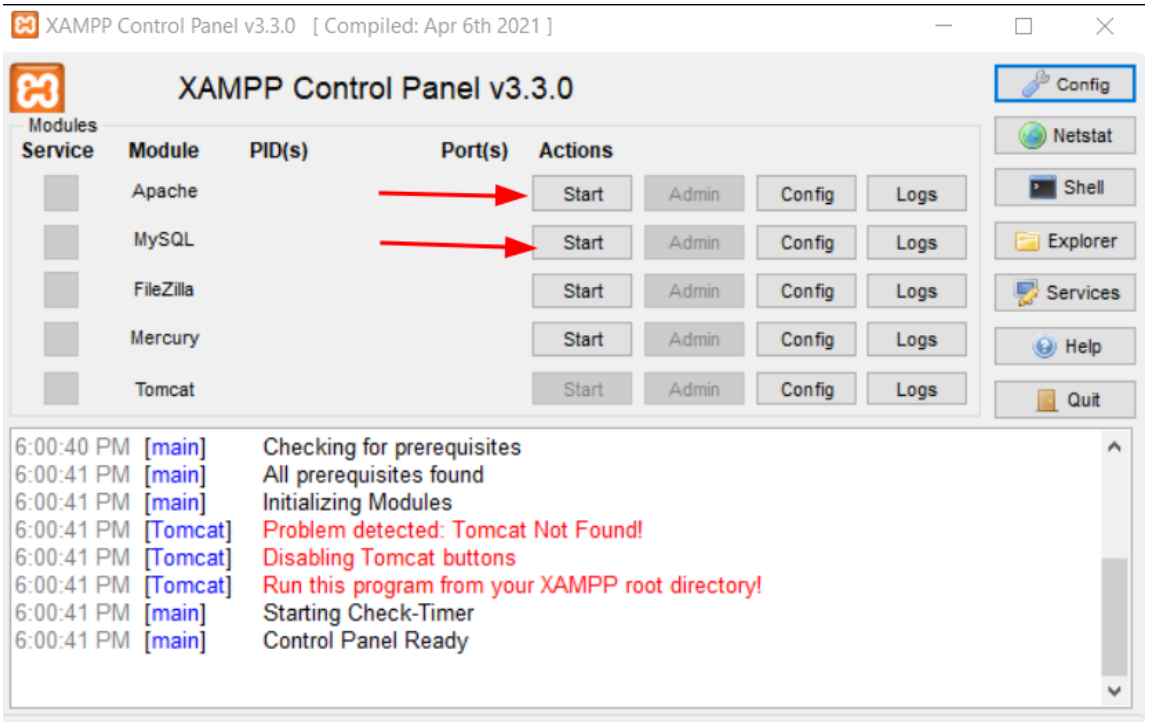

Figure 9.1.3.1 XAMPP Control Panel

You are only gonna need the Apache and MySQL for the web portal.

4. In order to display the pages you're going to need to store the information in a specific folder in the xampp folder: the htdocs folder as shown in figure 9.1.3.2

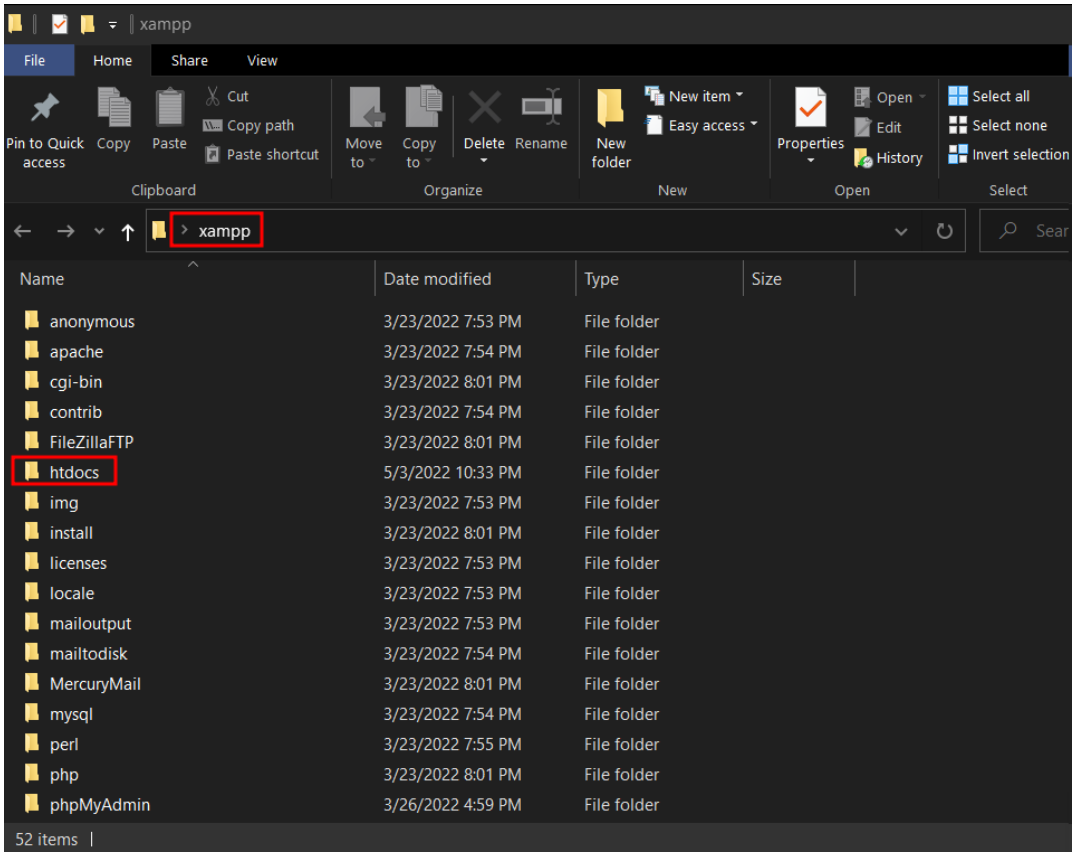

Figure 9.1.3.2 Folder Setup

5. Once you've saved your code in that folder you can see it by going to : http://localhost and as long as you have the index.html page set up it will display
6. You can then edit the code in any of your preferred IDEs like vim, VSCode, or Atom it only matters that the code is saved within that htdocs folder

9.1.4 Production Cycle

The first thing that was done in the project was that each module was developed in parallel but completely separate from each other. There were really four major modules that were worked on: graphical user interface, database, core engine, and the web portal. Initially each member was in charge of a single section and then helped the other parts as needed. This general strategy was used until after the technical prototype. After that the team started working closely together in order to integrate everything together into a working system. The general flow of work went as follows:

1. Create a feature
2. Test the feature
3. Test the feature with other needed parts and team members
4. Fix bugs found during testing
5. Do it all over again

This process allowed for general bugs to be found quickly and fixed. Integration testing also was an as we go basis.

10.0 References

JabberJack

[1] Miklosik, Andrej & Evans, Nina & Qureshi, Athar. (2021). The Use of Chatbots in Digital Business Transformation: A Systematic Literature Review. IEEE Access. 9. 106530-106539. 10.1109/ACCESS.2021.3100885.

[2] NAU. 2021.(2021). Retrieved November 3, 2021 from https://in.nau.edu/campus-health-services/

[3] NAU. 2021. Facts and stats. (2021). Retrieved November 3, 2021 from https://nau.edu/about/facts-and-stats/

[4] Rooein, Donya. (2019). Data-Driven Edu Chatbots. 46-49. 10.1145/3308560.3314191.

[5] Smilijanic Stasha. 2021. Amazon Alexa Statistics, facts, and Trends. (February 2021). Retrieved November 16, 2021 from https://policyadvice.net/insurance/insights/amazon-alexa-statistics/