**Artifact One:**

**Intro: 00:00**

Hello! My name is Kristie O’Brien and I am a senior in my Computer Science Bachelors Degree here at SNHU. Today we are going to be going over the three articles I am planning to enhance for my ePortfolio. Each of these artifacts were once assignments that I worked on in past classes in various stages of my academic journey.

**Artifact One: 00:18**

First up is artifact 1, which aligns with the Software Design and Engineering Category. This artifact is actually one of my first computer science assignments from all the way back in IT 140 which was also one of my first ever computer science classes. As such, I am very excited to be revisiting this assignment as I had a lot of fun working on it back in the day and to this day it remains one of my favorite coding assignments, even though I have gotten to work on many cool ones since then.

**Artifact One Structure: 00:46**

Let’s begin by taking a look at this code in terms of structure. First and foremost, I am happy to say that this code does in fact function as expected based on assignment guidelines back from IT 140 when I was taking it. It even had some extra features that were not required for the assignment, but I had some extra time back then and wanted to see if I could make those additional functions work as a new student.

The goal of this code is to be a text based game for the user where the user is interacting with a world based on text commands. So, for the context of this story, the map is a space ship, the Intrepid Archivist, and you are a researcher or maybe space archaeologist or something who has been exploring the stars in their favorite starship, but Oh No! Your arch enemy, the evil scientist has tried to sabotage your work by unleashing a stow away evil robot on your ship! And it is now up to you, the player to defeat the robot. The goal is to collect six items found throughout the map and then make your way to the room with the evil robot in order to defeat it. Each room has an item except for the room the player begins in, and then one room has the evil robot. If the player enters the room with the evil robot without all the required items, the robot wins, if the player enters the room with all the required items, the player wins. Either way, entering that room ends the game. It should be noted in this version of the game, the player cannot see the map so they have no way of knowing at the start where each of the rooms are in relation to each other, which adds an extra layer of difficulty.

So let us start taking a look at the code itself. There is not a specific coding standard that is being followed at this stage as this was a very early class. However, some standard conventions are followed such as white space between lines, indenting code within functions, and how the variables and function names are capitalized or not. This can be observed in the code. There are also several comments to explain the functionality, though we will look at these a little more in depth later.

Considering this was one of my first coding assignments, I would say this code is relatively well structures and it does maintain a consistent style. Naming conventions follow similar patterns as do indentation, spacing, and error handling forms. I do think there could be some potential ways to improve structure at this point in my learning. For example, all of the code does exist in this one dot py file. This could potentially be broken into header files or even some more detailed data structures. However, it should be noted that this is a pretty simple game. It is less then 200 lines of code and I have found that if it is less than that, it sometimes makes the code unnecessarily complicated to break it out into multiple headers which could potentially cause increased run times depending on how those are handled. So, as it stands, it is probably okay to keep the code in its current form. But if this code became much more complex to include combat or different maps, or even multiple players, it may be better to look into breaking this code out further for security, maintainability, and functionality purposes.

Having reviewed the code, I do not currently see any unneeded or uncalled for procedures and all of the code should be reachable, even if some of it is only reached in the case of the user entering the wrong input. Also, there are not any left over stubs of code or test routines that I was using back in the day to test the code.

This code already heavily relies on dictionary functionality and I have broken it up mostly into functions. I think in the game loop, there is likely more potential for making some of this code broken up into functions. But again, this could create unexpected behavior if not done properly with such a simple code and may in fact introduce counterintuitive complexity. It may be possible to have a some of the function calls, such as player status and how it interacts with the while loop be a little more concise. Sometimes it does appear that these calls can get repetitive. However, as a whole the code seems to follow the principles of unique functionality when possible.

Most of the data here is related to the map information, such as rooms, items in rooms, and which rooms are connected in the map. A dictionary in this case makes sense as this is a relatively small map with not too many items. Therefore, a dictionary is very efficient with no lags or issues due to storage in the run time. However, if the world were to be made bigger, with multiple levels or significantly more items, it may be advised to move away from a dictionary to another type of data structure.

**Artifact One Variables: 11:47**

Variable names tend to follow a string pattern as items within a dictionary. The only overly long module here is the while loop for the game, as discussed a little earlier, there are potentially ways to break this up. For instance, the end game could have the check functionality for items in the inventory in its own function, as could the commands for getting verses going in the command split. However, doing so may break down functionality and allow more errors as different variables would likely need to be passed between these functions. As such, for a simple code like this, keeping the while loop as is presents less unexpected behavior.

**Artifact One Run the Code: 15:09**

**Artifact One Documentation: 22:11**

Now let’s talk documentation. The documentation for this code is…shall we say extensive. The reason why is that this was one of my first ever coding assignments in college and as such, I wasn’t really sure of everything I was doing. One way I learn is to see if I can explain a concept back to someone else. If I can I find I often have learned something successfully. So I left myself a lot of comments to one see if I understood what I was doing and two to have as notes to reference later. I would not say at the moment that these comments are easy to maintain. Most are indented a little strangely, which I get why I did at the time, but in context now is not really the best option. Also, with how extensive they are, the code formatting can seem a little strange. As such, I would definitely condense these to code blocks and change their formatting to make reading and writing them easier. The comments are consistent with the code though, however, variable names could be better represented in these comments. For example, on line 139, instead of saying current room, with no line in between, it would be better to match the variable naming convention with the line between current and room. This is a consistent difference seen across applicable variable names between comments and codes that should be fixed.

Speaking of variable names, I tried very hard in this code to give really clear variable names as that also helped me as an early coder keep track of what I was doing in the code. So I would say all variable names are properly defined, meaningful, consistent, and clear. For example, current room is the room the player is currently in, user command is the command given by the user, and user input is what the user typed before it was split. Though, now that I look at it, it might have been better to name user command, split user input or something like that for clarity.

The type consistence I was supposed to keep for this class was consistently lower case. However, I find in more real world applications, it helps to have functions with upper cases and variables with lower cases. So while this followed course conventions, it might be worth changing the function names to upper case for clarity. Also, all the variables are also used here, there should not be any unused variables.

**Artifact One Arithmetic Operations: 25:55**

The only real math done here in this code is comparing whether the user has six items in their inventory or not. So the program will check the length of the inventory to see if it is exactly six or less than six. As six is a whole number, there were no issues comparing equality and it functions as expected.

**Artifact One Loops and Branches: 26:38**

There is one overarching loop that controls the game, which I usually call my game loop here. It is the while loop that will continue looping until the game is either lost or won or if the user chooses to exit the game themselves. My goal here was to never have an endless loop situation. The user should be able to exit gracefully when they are done with the game and the continuation of the loop should be seamless. This is reflected in the code with break statements, like those seen on line 108, and continue statements seen throughout the loop such as line 134, line 145, and line 155. These continue statements happen when a condition is being check for, such as error handling or collecting a new item. The break statements happen at the end of the game as once the end game is entered, the user cannot reenter the game loop.

All common cases are tested first in IF and ELSEIF chains. As we can see, from lines 127 to lines 151, if and else statements are used to validate user input and handle it accordingly. The user must enter very specific commands, and if they don’t the code must tell them what happened. These conditional statements do just that.

How the user enters user input is also error handled here. This specifically can be seen on lines 151, 157, and 113.

Which element of the dictionary that is accessed is also handled in depth in this code. That is why the user entered data is split on line 111. This allows the code to understand early if the user is trying to enter a get or go command to either retrieve an item or move to another room. This allows for early error detection and access to the proper information.

**Artifact One Defensive Programming: 29:44**

Memory then is allocated in condition in terms of location, with only inventory being stored. However, the inventory is set back to zero at the start of a new game, as can be seen on line 74.

**Artifact One Enhancements: 30:14**

This is the code as it currently stands, but what about planned enhancements? This code is a very good, basic text based game, but there are some things that can be done to make it better. Aside from cleaning up comments and potentially changing some variable names, this code could be made substantially more complex. One thing I’ve been wanting to do with this code is add in difficulty levels for the player. For example, right now, the player cannot see the map. It would be nice to potentially have the option for the player to select their difficulty level on game start and then select easy or hard for their level. This would likely be done before the game loop somewhere after the instructions on line 97. Additionally, for added difficulty, I was planning to add in additional items. These items could then be combined within the inventory to create new items that would then be able to counter balance how many of what kinds of items the user needs. For example, their could be harmful items, such as a corrupted cypher item that would actually detract from how many items the user has which would require them to have more items to defeat the villain. These items would be added to the dictionary starting on line 11, with logic to handle how many items are used and needed within the game loop.

The above described enhancements directly relate to my skills in problem solving in computer science by using different logic and error handling, while also implementing additional logic to add and remove items from an inventory. This corresponds with Course Outcome 1 which states : “Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision-making in the field of computer science.” By having different levels of difficulty, my code will be more accessible to a wider audience and user group. Additionally, adding different levels may better meet user needs and expectations. By polishing my code and refining it to better incorporate different player elements, my skills will also align with Course Outcome 4 “Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.” As my code will involve the usage of innovative techniques as well as various tools such as increased usage of dictionaries and loops to accomplish enhanced code tasks.

**Artifact 2**

**Artifact Two: 35:15**

Okay, so now let us take a look at Artifact 2, which aligns with the data structures and algorithm category for my ePortfolio. This code was my final project in my CS 300 class here at SNHU, and the goal was to implement data structures and work through those data structures with different algorithms or search abilities to interact with the data structure. As with the previous Artifact, this artifact as we see it here is completely unchanged from its original form back when I submitted this as my final project in CS 300. Right from the start, we see some housekeeping comments as I like to call them. These detail my name, the name of the class, the data structure I used, and so on. As we can see, the purpose of this code is to read in data from a file, parse it, store it in a data structure, print alphanumerically, and find specific course information and I used the vector data structure to accomplish this.

**Artifact Two Structure: 36:24**

Let us begin by looking at the structure here. This code was again developed in a pretty early class in my degree so there wasn’t really a rigid structure that I was working with. I did maintain some style guides though. As we can see I tried to use relatively consistent white space throughout, proper indentations, and so on. However, looking at this now, my white space isn’t the most consistent in the world and could definitely be cleaned up to make this code look a bit better and make it more maintainable in the long run. The structure here is not terrible either for an early code. I group print statements together, break out loops, sorting elements, and functions, naming conventions and new lines in print statements are relatively uniform. You can actually see an improvement in this code that I was talking a little bit about in the last artifact. As we can see in the scope of function names, like DisplayMenu up here at the top, the function names use capital letters while variables begin with a lower case, such as userInput. Again, this style could be cleaned up so that white spacing is more consistent, but the style as it stands is relatively consistent with itself.

Overall structure seeks to follow object oriented programming in as much as my understanding of object oriented programming was a bit rough at this stage. However, you can see that the different actions taken, such as processing user input, finding course information, and so on are split into their own functions. This helps to keep the code modularized for easier maintainability, debugging, and readability. By following object oriented programming, the code tends to follow higher security standards as well as there is less a chance of one corrupt function cascading its corruption to another function. However, security was not the main focus of this assignment.

None of the code is currently unreachable. The main reason for this is that the code is outlined to run based off of user input. As such, as long as the user inputs the proper information to access what they want to access, there should be no code that is unreachable. This being said, there does appear to be some duplicated logic now that I am looking at this between UserRunTwo and UserRun. Both are similar in how they handle the menu logic and it would be better if this menu logic was consolidated as, as it is right now, the logic is a bit confusing and having it repeated could introduce some strange behavior depending on the circumstances. So having that duplicated behavior is not best practices. However, there are no leftover test cases or code stubs for testing out the code and all the code that exists is meant to be functional.

Looking at handling the UserInputToUpper function, I am wondering now why I didn’t just use the C++ toUpper functionality. This code works just fine, it just might leverage the capabilities of the programming language better to have had this as toUpper rather than transform. But that is more of a stylistic thing and I did work on this several years ago so certain functionality may not have been as intuitive or available back then.

Storage use is relatively efficient. However, I did have the prereqList listed here as a class member in OpenFile, but the prereqList is only used per each course not necessarily the whole. So it might be better to make prereqList local to the OpenFile function. This being said, the code does work as expected with minimal lag time. Even so, the storage methods I used at this beginner stage could potentially be enhanced.

There aren’t any symbols used throughout this program. I primarily use hardcoded strings as numbers 1 through 9 to work through the menu and have the user interact with the code.

As discussed a little bit, it is worth noting again that the modularized functions UserRun and UserTwo do repeat the logic for the menu. This makes the modules excessively complex and they should be restructured to simplify the menu logic to one function if possible and where applicable. I could see this being condescend potentially within UserRun, taking the menu logic out of the hands of UserTwo for example.

Another thing to note, this code is pretty long and exists primarily in one source file. I do have a header file here, but it is more about defining functions and working through a few support functions. If I was going for a more object oriented programing style here, I would likely look at splitting the source file into more header files if the code were to grow substantially longer. As it stands now, this code is not lengthy to the point of being out of hand as it is a relatively simple code. That being said, if this were to be developed to work as a larger or more robust database for courses, then changing the code structure to split it between more files may be helpful.

**Artifact Two Documentation: 47:40**

Now let’s talk about documentation. It amazes me to compare my comments from this artifact to the previous one discussed earlier. The comments here are still extensive, going line by line, however, at this point I had mostly done away with those bulky paragraph inline comments that messed with the formatting and caused the code to look stranger. This makes for an easier to maintain commenting style. However, the comments could be reduced in number and there could be fewer inline comments at this point and more block comments prior to blocks of code. This would make the comments more readable and easier to maintain. Again, the reason for the extensive commenting was that this was a very new way of coding for me and a relatively early class in my degree, so I wanted to make sure I knew what I was doing. The comments helped me to keep track of that. However, now that I have more experience, these comments could be cleaned up a bit.

For the most part, the comments also seem to be consistent with the code. In the comments, I refer to the functions as they are written in the code rather than as they might be written in a paper or spoken allowed.

**Artifact Two Variables: 49:14**

I do believe the variables are properly defined with meaningful, consistent, and clear names. For instance, userInput relates to user input, getPrerequisites relates to retrieving the prerequisites, and so on. One that I might change would be the bool variable called flag. This is not a bad name as this variable does act as a flag. But perhaps a more descriptive name would be something like course found or course alert as it tells what the flag is doing.

Throughout the code, I properly type the variables as strings, bools, integers, and so on where applicable. As such, the variables are properly typed. There also do not appear to be an unused or redundant variables.

**Artifact Two Arithmetic Operations: 50:52**

This code does not use much arithmetic for comparison. As such, there is not an instance where floating point numbers are compared for equality. As such, there is also not a cause for concern with rounding errors as this code does not directly deal with rounding numbers. Additionally, this code does not deal with any additional math such as division, multiplication, addition, or subtraction. As such, common coding pitfalls that may arise from these are not applicable here.

**Artifact Two Loops and Branches: 51:24**

The loops that exist, and there are a few, are implemented such that they are logical, complete, correct, and properly nested. A key way I can see this in my code is there should never be an endless loop situation. The loops continue looping and act based on user input. For example, an option for the user to exit the loop is given by the user selecting 9 as an option. This option is clearly communicated to the user via the menu. If the user does this, a break statement is called which allows the loop to end seamlessly. Meanwhile, the various continue statements, allow the loop to continue rather than getting stuck with invalid user entered input. Additionally, one could view the loops here like a game loop. The overall while loop should continue looping as long as the user does not exit the loop themselves.

**Artifact Two Code Run: 53:48**

Now that we are talking in more detail about the loops, let us take a look at how the code functions. When I start up the code, we are presented with this menu. From here, I can enter input as directed. See if I try to enter 2 before I have loaded the data by entering 1, then I am presented with a message prompting the user to load the data. After I do that, you can now see how I am able to load the data. But what if I entered an invalid number like 0? A message then appears which tells the user that that is not valid input. In the code then, we can see how this corresponds to most common cases, as well as additional cases, are covered by IF ELSEIF or ELSE blocks. These handle correct input such as did the user enter a number between 1 and 9 and also handle invalid inputs.

**Defensive Programming: 56:04**

In this code, files are tested to see if they properly opened to avoid unexpected behavior with unopened or improperly opened files and data writing. Also, courses without prerequisites are tested for accordingly. This prevents unexpected behavior such as populating a prerequisite list that is invalid or does not exist. Meanwhile only the data that the user requests to be called is called. This prevents incorrect data from being operated on.

**Artifact Two Enhancements: 57:21**

Now that we have looked at the code in more detail, let’s talk about the specifics of enhancements. As this article is meant to correspond to enhancements related to data structures and algorithms, I chose to work to make this code more efficient. Currently, my code works with an O(n) complexity. This can be improved to an O(1) complexity. By using an unordered map to look up the courses in the data structure. As of looking at this code in the moment, I am currently planning then to implement this change primarily in the FindCourseInfo function. This function uses a for loop, which increases the complexity. By using an unordered map, the map is populated once when the file is loaded and then it is referenced when needed to print courses rather than looping through each course every time.

By completing this enhancement, I will show the specific skill of understanding different runtime complexities, optimizing runtime complexities, being able to implement multiple methods to search through a data structure from both the original and the new method, problem solving for computer science problems, and code formatting for proper data structure and algorithmic interactions. As such, I believe these skills best correspond to Course Outcome 3 which states “Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.” And Course Outcome 4 which states “Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.”.

These course outcomes are specifically related to understanding the design benefits and issues that can arise from different design choices in computer science. By making my code more efficient, I will be showcasing how I understand and work with this tradeoff for more efficient code with potentially less intuitive code design for those that may be more familiar with a for loop rather than an unordered map. Meanwhile, I will be showing how I use innovative techniques and skills for computing practices by implementing unordered maps to better interact with the code and the data such that the code is more maintainable, able to be debugged, and enchantable for potential future expansion.

**Artifact 3**

**Artifact Three Intro: 1:01:16**

Okay, so now let us take a look at Artifact 3. Artifact 3 is probably the most recent of all my artifacts that I have presented here so far and it was from my CS 360 class where the main focus was app development. As with the others, this artifact is unchanged from when I submitted it as my final project for CS 360. The purpose of it was to create an app in Android Studio, that is the IDE I am using here, to function as an inventory app. So the user can sign up for the app, login to their specific inventory, add items, subtract items, increase the number of items, and overall maintain their inventory within the app. In fact, we are going to change things up a little bit.

**Artifact Three Code Run: 1:02:28**

Let’s start this code review with showing the functionality as this is a pretty big grouping of code. First off, we come to the login page, where a user can either enter their credentials to begin with or sign up. For the sake of showing the majority of the functionality I will go ahead and sign up with new user login credentials. So I will type in some credentials here, select sign up. And now I can login with those credentials. From there, I am taken to this inventory screen. As you can see on start up it is empty because this is a new inventory and the user hasn’t input anything yet. We can go ahead and add an item by selecting the add item button. And this allows me to enter an item name and amount. Now we can see 12 oranges have been added and I can increase or decrease this number. I can then add another item and then delete it. I can also add SMS notifications that will allow the user to receive a text message if an item in the inventory reaches zero. And then I can log out of the app. So that is a general run down of how the app works and functions.

**Artifact Three Structure: 1:06:27**

As you can see on the side bar here this is a very lengthy app. We have multiple classes to complete actions, such as adding a new item, inventory maintenance, the main activity for the app which basically calls everything and runs it, the SMS notification functionality, and the user database. I also have all the associated layout functions. So all of this code functions as expected as laid out in the course specifics for the CS 360 class at the time that I took it.

As a whole, this follows some of the principles of object oriented programing in terms of its structure. For instance, we have public classes that extend different activities in order to retrieve information from those activities or classes as the case may be here. Separate functionalities are split into their own classes, and proper principles of public and private variables are adhered to. Additionally, the style is kept consistent throughout. I adhere to similar naming structures, spacing, and indentations throughout. This is a much later code in my degree so it has been really fascinating to me to see how much my use of style has changed over time in just these three artifacts.

Looking at this code big picture, there are not any uncalled for or unneeded procedures or unreachable code. The code functions to handle errors, handle input, and handle expected behavior and unexpected behavior respectively. Each call is specific to a function and therefore there are no uncalled for or unneeded elements here.

There are also no leftover code or sub routines that were left in here for testing. And that is definitely something I sought to double check as, if I recall correctly, I encountered a lot of errors in this work and as such, needed to perform a lot of tests to make sure I knew where those errors were coming from and how to fix them so that I could fix them properly.

It should be noted that within the entirety of this project, there is going to be a specific part of it that has our focus, so my review is applicable to all of the code as one portion of code follows the conventions of the others. However, this being said, I am going to be looking primarily at the portion that needs the most improvement and that is the User Database class.

Some of these methods, such as adding a user, adding an item, or updating an item amount can appear a bit redundant. As such, an improvement that might be helpful here would be to add helper methods instead of hard coding these elements. However, helper methods could lead to some interaction between the methods that is unintended so such an improvement should be carefully considered. As it stands, I do believe that how the code is now is pretty intuitive in terms of its clear what does what. But if this were to be expanded further, I would say helper methods could be very beneficial.

My storage methods are pretty efficient here too, and this follows throughout the code, I am using integers for amounts and user ids with text data types for text fields like usernames and passwords. These are good storage data types for these methods. However, how I am storing the usernames and passwords is a plain text and this might be a bit problematic in the future. So this is just something to consider and will be discussed in greater detail in a little bit.

I primarily use constants or string constants here rather than magic numbers. For instance, I use COL\_ID and COL\_USERNAME to store constant representations of specific data. This also helps to avoid overlap in data between users.

Most of the modules are very finetuned to only handle what they are meant to. For instance, adding a user’s only purpose is to add a user, same for adding an item, and so on. In theory, I could split this user database into two databases though, such that everything for user credentials is handled in a separate file and everything for items is handled in a separate file. However, for a smaller program like this, having it all in one file made the most sense for me at the time in terms of debugging.

**Artifact Three Documentation: 1:18:16**

My documentation style is still primarily inline comments at this time with lots of detail on code functionality. I normally would not comment to this extend at this point in my degree. However, Android Studio and coding for apps was something I was very much not familiar with. As such, I left a lot more detail in the comments then is probably necessary. I would likely limit in line comments and focus more on block comments. This could help increase the readability of the comments as well as their maintainability. This being said, the comments are consistent with the code, with comments calling variable names as they are written in the code in the comments and reflecting what the code does well.

**Artifact Three Variables: 1:20:03**

The variables I believe are properly named and defined as the names reflect well what they accomplish. For instance, username is reflecting that this data type is for a username, AddItem is a function to add an item, and so on and so forth. This is a style I sought to keep throughout the code as it made it easier for me to understand what different functions, classes, and variables did. This was essential to me in the debugging process of this project. As such, there are also no unused variables here. One thing I do love about this IDE, is that it made it very clear when something was unused by changing the color of it. Because towards the end of working on this assignment, I recall I did have quite a few unused variables leftover from my test cases. So the IDE made it easy to find these and weed through them.

**Artifact Three Arithmetic: 1:22:05**

This inventory app does not allow for half an item to be in the inventory so there are never any floating point number comparisons here and there are also no rounding errors as the only math done here really is just either addition or subtraction. The code does avoid additions and subtractions on numbers with greatly different magnitudes as well because if a user is increasing an item amount, they are just incrementing it by one, and if they are decreasing an item amount, then they are just decreasing it by one. So addition and subtraction are in increments or decrements of 1.

**Artifact Three Loops and Branches: 1:23:55**

Loops are not really applicable in this code as most, if not all, the searches are handled through SQL queries. So there are really no loops that I can recall, or now that I am looking at this again, see, in this code. However, I do use a lot of IF chains throughout, such as on line 86. These check to see if certain conditions are met and then decide what to do with them if they are not. For example, as can be seen on line 86, this asks if there is more data still to search through and reacts accordingly. I also use in various places throughout this code, such as in the InventoryActivity class, use try catch blocks for error handling, especially for working with the SMS manager in java.

I could have some additional testing here on cursor validity. For example, there is not a check here for an empty username or password. However, such an action is prevented as a user is prompted to add all fields should they try to sign up or login with an empty username or password. Output variables to the database are assigned to their table accordingly be it a username, password, or item. The correct data is handled such that only the logged in user can see their data.

**Artifact Three: Enhancements: 1:28:21**

Now, this takes a look at the existing code, but what about some specific enhancements? One thing caught my eye almost right away after reading through this code again and having just completed my secure programming class here at SNHU. And this is the fact that there is no security on my usernames or passwords within the database. This could potentially lead to a security breach on my database, leaving the database vulnerable to attacks. As such, my main plan for enhancing this database is to add security on the usernames and passwords. I plan to add SHA-256 hashed elements for the password storage, ensure better data integrity as right now, I am not ensuring that data is in and of itself unique, and making sure my code is secure from SQL injections. This means ensuring I don’t have any tautologies in my searches such as 1=1. My plan specifically to combat the SQL Injections is to use pattern recognition with REGEX to detect suspicious queries and reject them.

I believe these enhancements show my skills in understanding the importance of security to databases by utilizing encryptions, unique data, and protection against SQL Injections. Specifically protecting the code from SQL Injections was chosen because this code relies heavily on SQL queries. As such, SQL queries are a point of vulnerability for this code. Therefore, it is important to protect it from this. As such, and by utilizing various encryption tools and capabilities as well as libraries and functions for handling REGEX pattern recognition, I am showing how I am capable of using a wide range of tools to solve complex computational security and database issues. Therefore, I believe these skills align with the course outcome 5 which states “Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.” And course outcome 4 which states: “Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.”. These course outcomes are both tied to problem solving, using multiple ways to solve a problem, the importance of high quality code, and code security. As I plan to implement multiple security standards to increase security in my database, I am showing a mindset for secure coding. By using the various tools, libraries, and different database security techniques, I am showing how I am able to implement computer solutions in such a way that is valuable to the finished product and the industry.

I am excited to get to work on all of these artifacts and creating my ePortfolio. Thank you for watching my code review!

Course Outcomes:

1. Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision-making in the field of computer science.
2. Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.
3. Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.
4. Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.
5. Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.