The following is my entire planned list of enhancements for this artifact, as previously included in my updated CS499 Final project plan (Module 1 assignment).

Artifact Enhancements #3 – Database - Enhancement Requirements List:

Enhancement 18: Create a database with separate tables for Client and Employee data.

Enhancement 19: Develop a database manager trait with needed CRUD operations for both Client and Employee tables.

Enhancement 20: Develop a client manager trait to handle client-related operations both locally and in the database.

Enhancement 21: Develop an employee manager trait to handle employee-related operations both locally and in the database.

Enhancement 22: Implement functionality to update employee-client pairings both in the database and local storage.

Enhancement 23: Add capability to update client service selections in both the database and local storage.

Enhancement 24: Implement methods to retrieve single client objects by client_id (pull from local AVL tree).

Enhancement 25: Develop functionality to retrieve client lists by employee_id (using the local hash map).

Enhancement 26: Create a method to retrieve all Clients.

Enhancement 27: Implement a transaction system to manage local and remote system operations. Utilize it to encapsulate client & employee operations to ensure that data remains consistent between remote & local storage.

1. Briefly describe the artifact. What is it? When was it created?

The artifact selected for this project is the same artifact that I used for my artifact 1 and 2 enhancements. This was the final project from my CS410: Reverse engineering course. It is a client management application that I was asked to build for SNHU Financial Investing Firm. We were asked to take the provided binary executable, disassemble it into assembly, then decompile it back into C++. We were then told to use our knowledge of security to rewrite the application using industry standard best practices. I originally rewrote and submitted the application in June of 2024 for my final project in the course.

2. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item?

My reasoning for selecting this artifact is primarily the same as for the previous enhancements. This project was written at a point when my knowledge of computer science I feel, was at it's greatest within the program. My time spent in CS410 was incredibly rewarding, and I feel strongly that my final project for this course displayed a strong understanding of computer science principles, and that it was some of my best work in the program. In the initial project, were given the freedom to redesign the application as we saw fit, we were just asked to ensure that we removed any security flaws from it. The application that I was left with after CS410 was in a good position, but there were a number of features that I noted at the time that I would like to get around to implementing. With these artifact enhancements, I have had the opportunity to really dig deeply into the application that I wanted to turn it into, and I am happy so far with how everything turned out.

<u>What specific components of the artifact showcase your skills and abilities in software development?</u>

There are several noteworthy components within the third stage of artifact enhancements that I think really highlight and showcase my abilities and growth within software development. The first components I would like to focus on are my client and employee handlers. These structs and their implementations are both within the operation_handlers.rs module, and they are at the center of my application. These two handling structures manage all operations within the application, in regards to the Employee or Client instances. Either one of these components is used to pass dependencies for the functional operations within the application to the various corners of the application. For instance, with the authentication process for the application, an instance of the EmployeeHandler is used to retrieve the password hash that is stored within the database for a specific employee. It is retrieved to be used within the authenticate function. The menu system handles / directs the application to various client related operations depending on the provided user input, and in order to perform these operations, the menu system relies on an instance of the ClientHandler. These handlers are responsible for initiating transactions, and then for carrying out modifications to the data stored both in the database, and locally within that transaction. The handler components are also a means by which to initially retrieve data from the database, and to populate local data structures. The transactions that I am speaking of are the next specific components that I think are worthy of covering. When I was researching and looking for approaches to properly handling operations that affect multiple data sources / storage locations, I came across the concept. A transaction is a means by which to encapsulate a series of actions that are performed using data sources. We can use them to specify how we want the operations to be carried out, as in my case, if I am making a modification to one data storage location, I need all locations containing that data to also be modified in the exact same way. My transactions are initiated before the first change / modification is made, the attempts are made to modify the data stored within the different sources, and then if all operations are successful, the changes are then committed. If any of the operations fail, such as the data in the database failing to accept the changes, but the operation was successful locally, then the transaction would not be committed, and the changes would be rolled back. The inclusion of this feature into my application is an enormous improvement over more simplistic approaches I could have used, and I believe showcases my ability to learn and implement complicated methods successfully into my code base. The last component that I think this artifact showcases, is my successful integration of a remote database into an application that previously relied on solely local data structures. Showing that I am capable of successfully integrating a database I believe is a significant step in developing your skills as a programmer. The database that I implemented will correctly populate the application at runtime, and it greatly expands the work that can be carried out within my application. Data is saved between sessions without the need to rely on file output for storage. The application can easily be built from source on any machine, and after providing valid authentication details, can quickly have access to the updated system data.

How was the artifact improved?

In my original application, there was a single hashed password that was used to provide access to the system. There were not multiple accounts for different employees. The application processes were structured similarly for the operations that could be performed on instances of the Client class objects, via a client management class instance. In my final version, all operations for the client struct are managed with a similar ClientHandler implementation. Instances of my Employee struct are similarly managed with the EmployeeHandler implementation. A great deal of added functionality was created in the system with the inclusion of the Employee data model, and its dependency / operations handler.

In addition to the employee and client handlers, I added the MySqlDatabase integration object which is used to establish a connection to the remote MySQL database, and a DatabaseManager trait / interface that is implemented for the database object to handle queries / operations.

With my addition of the database system, I also needed to provide a way to maintain consistency of data across the different storage mediums. This is the primary reason why I implemented my transaction system. It helps to ensure that any data additions, modifications, or removals remain consistency between local data structures, and the remote database source.

The following are the enhancements / improvements that I made to my final version of the application:

Enhancement 18: Create a database with separate tables for Client and Employee data.

The first of the enhancements that I set out to implement within my application involve the creation of the remote database. I ended up using DigitalOcean's MySQL database hosting services, as my initial idea to use AWS ended up being a much more involved process than I expected. I rented one of their MySQL relational databases, which provided me with all of the necessary connection details. I used the following commands to set up the initial database using the database management application, DBeaver.

```
CREATE TABLE employees (
employee_id INT AUTO_INCREMENT PRIMARY KEY,
employee_name VARCHAR(75) NOT NULL,
hashed_password VARCHAR(150) NOT NULL
);

CREATE TABLE clients (
client_id INT AUTO_INCREMENT PRIMARY KEY,
client_name VARCHAR(75) NOT NULL,
client_service INT NOT NULL,
assigned_employee INT,
FOREIGN KEY (assigned_employee)
REFERENCES employees(employee_id)
);
```

This set up both the Employee, and Client tables each with their desired attributes. I was then able to validate that the tables were correctly implemented with the SHOW TABLES command, and the DESCRIBE employees and DESCRIBE clients commands. After these initial stages, my database was ready to be seeded with test data for my application. To create the clients list, I created a new excel / csv file, and I used the fake names generator at this address: https://homepage.net/name_generator/. Once I had the csv file properly formatted, I created the employee database seed function using the employee account details provided within the auth_data.csv file. Once I had seeded the initial list of employees to the database, I finished setting up my client list file, with a relatively random employee assignments. I then used DBeaver to seed the now formatted list of client names that contained service choices, and assigned employee values. My database was then fully set up and contained all of the data needed to begin operating normally.

Any of the MySQL queries that would be needed to perform operations within the program could be accessed and executed via the implementation of the DatabaseManager trait for the MySQL database. The DatabaseManager trait is the interface that contains functions that perform each of the required query operations.

Enhancement 19: Develop a database manager trait with needed CRUD operations for both Client and Employee tables. (Implemented in the database.rs file, DatabaseManager trait/implementation of MySqlDatabase.)

As mentioned briefly in the previous enhancement, the DatabaseManager trait is the interface that actually provides the queries that are used to access / retrieve data from the database for the use in the application. This is done through the process of function delegation in Rust. What this process entails, is that the DatabaseManager trait defines the functions that every implementation of DatabaseManager should implement. This trait is then implemented for the MySqlDatabase struct, which is what contains the pool for the database connection. The DatabaseManager implementation does not itself handle the process of connecting to the database, only database based operations. The methods that are implemented for the MySqlDatabase then are used to delegate the actual work / operations that needs to be completed to the pool member of the MySqlDatabase implementation. This is one of the ways by which we can practice composition based design in Rust, and a means by which we can provide polymorphic behaviors.

Enhancement 20: Develop a client manager trait to handle client-related operations both locally and in the database. (Implemented with the struct/impl ClientHandler in operation_handlers.rs)

The approach that I ended up using for both enhancement 20, and enhancement 21 were the same. Initially I thought that I might write them both as interfaces / traits. This ended up not being the best approach, as there were a number of functions that I wanted to encapsulate within the handler, and I wanted the flexibility to be able to define which functions were public and available for external calls. Using a trait would mean that I am exposing all of my fields and methods. Instead, I utilized composition to include the implementation of a DatabaseManager object as a required field in both the client handler struct / implementation, and the employee handler struct and implementation. In addition to requiring that each handler have access to an implemented DatabaseManager object, I included an instance of the local data structures that would be needed in order too facilitate local program operations. The reason for using local data structures in combination with the database storage, is that the local data structures allow a much higher level of efficiency of operations. This combined with the best practice desire to reduce the quantity of queries on the database to the minimum necessary.

Enhancement 21: Develop an employee manager trait to handle employee-related operations both locally and in the database. (Implemented with the struct/impl EmployeeHandler in operation_handlers.rs)

This enhancement follows the same blueprint as used to create the ClientHandler, as mentioned in enhancement 20. I ended up utilizing a struct based, composition approach rather than a trait / interface approach, as that allowed me the greatest amount of control over the access to what could be considered sensitive data. Using a struct, and then implementing that struct directly with the needed dependencies allowed me to limit public access to only functions that required it. Because the struct is composed using an implementation of the DatabaseManager, it is given access to needed dependencies related to remote database operations. Any local operations would be performed using the data structures that are included in the struct definition, such as the stored_hashes structure that is used to lazily cache password hashes after the initial retrieval to limit the quantity of queries necessary to complete the authorization process.

Enhancement 22: Implement functionality to update employee-client pairings both in the database and local storage.

Added:

- In database.rs module: get_employees() and get_employee() functions added.
- In firm_models.rs module, the change_client_employee_pair() function was added.
- In operation_handlers.rs module: EmployeeHandler impl, get_employee() and is_valid_employee_id() functions added. Added a hashmap “stored_employees” to store

retrieved employee objects from db, the hashmap is checked first, and then the database is queried when attempting to locate a specific employee.

- In menu.rs module: change_client_employee_pair(), client_pairing_handler(), and get_new_pair_employee_id() functions were added.

This functionality was added by implementing a number of related functions across several different modules. These modules included the database.rs module, the menu.rs module, the firm_models.rs module, and the operation_handlers.rs module. For the database.rs module, the modifications were added with the addition of the get_employees() function, the get_employee() function, and the queries that both of these functions utilized to retrieve employee data from the employee table of the MySQL database. The functions that were added to the menu.rs module initially started out by duplicating the methods that I had used to implement the service change menu options. I adapted these copied functions to instead allow the retrieval of Client objects for the purpose of modifying the value that represents the employee that the client is paired. This is facilitated through the change_client_employee_pair() mutator function of the Client implementation in the firm_models.rs module. These modified Client object is then used to update both the AVL tree local data structure, and the database through the ClientHandler of the operation_handlers.rs module. The same update_client() function that is used for client service changes can also be used with the Client object that has had its employee client pairing changed. Additional functions that were added to the ClientHandler implementation to facilitate this process include the get_employee() function, and the is_valid_employee_id() function. An additional data structure, "stored_employees" was added to the EmployeeHandler struct definition. This additional hashmap is lazily loaded in a similar manner as the stored_hashes hashmap used for the authentication process. When the get_employee() function is called, the stored_employees hashmap is checked first, if the match is found there, then the database is not queried. When the target employee object has not yet been retrieved, then the database is then called, and if a match is found, then it is added to the local cache / hashmap, stored_employees.

In all, these modifications fit together to provide the added functionality of an additional menu option, one that provides another service to the user.

Enhancement 23: Add capability to update client service selections in both the database and local storage. (Implemented in the operation_handlers.rs module, using ClientHandler, & Transaction definitions.)

This functionality is accomplished with the help of the transaction system, as well as through utilizing the appropriate ClientHandler functions for the desired type of modifications to a Client object instance. I say desired type of modifications, because while initially I had planned to limit this to the modification of client service selections, a very similar process is used in conjunction with the update client function to modify the assigned employee pairing for a client instance.

Enhancement 24: Implement methods to retrieve single client objects by client_id (pull from local AVL tree). (Function named get_client() in the ClientHandler (operation_handlers.rs) module

This is accomplished by selecting the menu option to select a client for modifying (either their service, option 2, or their employee pairing, option 3). The get client function of the ClientHandler implementation is used to search through the AVL tree for a matching client ID value. It does this by using the user provided client_id value and the find() function of the local AVL tree to search for a match. If one exists, then it is returned and used in a match statement to select the Ok(Client) branch, the function may then proceed with modifications to the Client object if desired. Otherwise the error branch is taken and the relevant error is returned.

Enhancement 25: Develop functionality to retrieve client lists by employee_id (using the local hash map). (Implemented in operation_handlers.rs, ClientHandler, get_clients_for_employee() function.)

This enhancement is implemented using the hash map that is populated when the ClientHandler constructor function is first called. The constructor populates the local AVL tree with client data, and the client employee pairings hash map with the appropriate data. To retrieve this data, the display clients for employee menu option is selected, and the relevant employee id value is provided. This uses the employee id key value to locate the vector of client objects that is associated with the provided employee id. The user can then use the listed values as a means by which to selected and modify client objects through the other main menu options

Enhancement 26: Create a method to retrieve all Clients. (Named get_clients() in database.rs module. Called in constructor of ClientHandler.)

The method for retrieving all clients is first added into the database.rs module. This function includes a Select query that retrieves all of the necessary data to create each Client object, or each row from within the clients table of the MySQL database. Each row of the clients table is mapped to a new Client object, and is added to the clients vector, \`Vector<Client>\`. This vector containing all client objects from within the clients table is then the return value when the operations are successful. To utilize this function, and the vector of clients it returns, the constructor of the ClientHandler iterates through the vector of clients, and adds each of them to the local AVL tree that is used for the majority of operations within the application. During this process of adding each client object to our AVL tree, we are also adding each of the clients into a local hashmap that is used for outputting lists of clients by their associated employee ID.

Enhancement 27: Implement a transaction system to manage local and remote system operations. Utilize it to encapsulate client & employee operations to ensure that data remains consistent between remote & local storage.

Implemented in the operation_handlers.rs file, used in the employee/client handlers to add, modify, or remove data from both the local data structures, and the remote database. The transaction is written as a struct, one that contains a boolean member that represents the state of the transaction, whether or not it is completed. The second member of the transaction struct is a lifetime reference to the DatabaseManager implementation used for the application. The database trait implements a begin_transaction method that utilizes the MySQL query "START TRANSACTION". It also contains a function for committing the transaction, and for rolling back the transaction, using the "COMMIT" and "ROLLBACK" query respectively. The transaction implementation constructor calls the begin_transaction function to initiate the transaction whenever an operation is performed within the application that would need CRUD related functions. If the transaction operations are successful, then the commit function of the transaction is called. If an operation fails, and/or an error is generated from the process, then the we allow the drop function to be called, which rolls the changes back. The drop functionality is handled separately by implementeding the Drop trait for the Transaction struct. The conditions for calling drop also include instances where the transaction is no longer in scope. To illustrate how this might happen, I will reference the update_client function of the ClientHandler implementation. Each individual instruction / operation features the question mark operator at the end of the operation. This is so that when / if an error occurs, the error is backpropagated from the function immediately. The first operation is to create the transaction with the new keyword, assigning the transaction to "transaction". We then use the transaction, which contains the mutable database reference, to call the update_client method with the provided Client object. Then, using the ClientHandler self reference, we remove the existing client object that matches the provided Client's id from local storage. We then insert the updated Client into the local storage structure. The last operation is to use the transaction created within the scope of the function to call the commit function to commit the changes. If at any point during this function, an error is generated, and an operation fails, the function call will end, and the transaction will no longer be in scope, which results in the drop implementation being used. The transaction is then rolled back.

3. Did you meet the course objectives you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

There were two course objectives that I was initially setting out to meet with this stage of artifact enhancements.

The first of these objectives, is to show "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." I met this outcome by creating a working database, and data handling system for my application. I used the most recent techniques, and the up to date documentation for the newest version of MySQL that was available for use in the rust language. I believe that I was able to implement a robust data management system for my application that more than delivers on the features and value that I initially set out to create in my enhancements for this category.

The second objective was to display, "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." The process of determining what method would be the best approach for maintaining security in this application was particularly complicated without being formally taught it. It was something that required reading through the documentation, and stack overflow posts for feedback on what I felt would be the best way to handle sensitive information such as the secrets that are used to establish the database connection when the application runs. Obviously the one thing that I could not do was hard code the login credentials into the application anywhere, and that would also include not hard coding login authentication data for the application either. What I ended up settling on, was to use a configuration file and environmental variables to create the initial database connection and set up the pool. I implemented a similar approach for both the containerized docker built version of the application, and for the locally built application using rust's cargo commands.

The outcomes that I have mentioned accomplishing above are not new outcomes, but I believe that I have again met them with additional criteria over the course of completing my Artifact enhancement 3 assignment. I do however believe that with the addition of an explanation video, that I would be able to satisfy the criteria of additional outcomes, specifically the outcome, “design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.” Using my written documentation, along with visually presenting my the enhancements to my application, I will provide quality oral communications that highlight on the enhancements that I have made.

4. Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?

With the conclusion of the final enhancement component of my artifact, I am feeling more confident in myself, and in my ability to solve problems through coding. I think that it is safe to say that working on this project has challenged me in ways that I have yet to experience during my time learning programming and Computer Science principles thus far. I don't think I can overstate how much time and effort it took to actually get the database component of this application functioning in the desired way. I faced a number of challenges that required me to essentially completely change the way direction I was approaching the problem from. I had previously wondered what the process of authenticating and connecting to a database remotely would look like in an application, as my prior experience with them was limited to locally running databases. It was a significant undertaking to need to figure out how to handle secrets files, the various steps I needed to follow to ensure that I was not exposing them, committing them to git, or into remotely run docker images. I ended up with the application having two approaches to building from source successfully. The first was through my inclusion of a dockerfile that would correctly build the application and run it within a container. The second is using rust's own cargo build / cargo run commands. I also began the steps of hosting the application itself remotely on an application platform, and I was able to eventually get the application compiling on the application platform that I rented for it. As I did not provide the application with an interface by which to be accessed through that remote platform, it ends up just reloading the login / authorization prompt until it crashes. This remote platform would rely on the environment variables set with the host to establish the database connection instead of needing to provide a secrets file. The locally built options both rely on using the provided config.toml file in order to establish a database connection.

Challenges faced:

Of the enhancement sections that I worked on this semester, I think that the database implementation was by far the most tedious and time consuming. My implementations for it evolved and transformed a number of times over the semester. Most of these changes were isolated to the MySqlDatabase connection implementation, with the some additional functionality added to the operation_handlers.rs module. The most tedious process was attempting to get the implementation for the MySqlDatabase struct correctly connecting to the remote database. The vast majority of advice and implementation approaches that I found online were for very outdated MySQL crate versions. I ended up using an OptionsBuilder and using a configuration file / file path for the successful connection attempts. Apparently at some point in the last 6-months to a year, the methods/options used for setting up a connection has drastically changed. I ended up almost entirely having to rely on the documentation page for the most recent MySQL rust crate, as that was one of the only locations that I found that had implementations which were compatible with the updated MySQL rust crate. I spent the better part of two days working with the compiler and various different configuration attempts, before I was able to get things working. I then ended up having to almost completely rework my approach to allow me to use multiple different build approaches. My initial approach was only using Rust's cargo commands, but as I was attempting to get the application to a point where I could launch it on a remote application platform, I also needed to provide other optional connection approaches that would depend on how the application was built. At least with the application platform I rented space on, it required me to set environmental variables for the connection details, and to use docker to build the application in a container. I did get the application to a stable state, and was able to include a working option to build the application locally with docker in addition to the ability to build it normally with the cargo build command.

The majority of the other challenges that I faced involved the process of properly implementing error handling across the various modules, and ensuring that I was properly implementing my return values using the expected types. There were a few instances where I ended up needing to define an implementation within the errors.rs module to allow for the conversion of standard library types into my custom error types. I consider myself lucky as the majority of these errors the compiler provided very clear with messages that let me know that "the type being implemented / returned here does not meet the expected type of x". I was then able to implement the conversions from the one type to the type that I was attempting to use.

Useful Documentation Links Used in Coding:

- *argon2 - Rust. (n.d.). https://docs.rs/argon2/latest/argon2/*
- *How to write documentation - The rustdoc book*. (n.d.). https://doc.rust-lang.org/rustdoc/how-to-write-documentation.html
- *Introduction - The Rust Style Guide*. (n.d.). https://doc.rust-lang.org/nightly/style-guide/
- *Error handling - the Rust programming language*. (n.d.). https://doc.rust-lang.org/book/ch09-00-error-handling.html
- *macro_rules! - Rust By Example*. (n.d.). https://doc.rust-lang.org/rust-by-example/macros.html
- *Attributes - the rust reference*. (n.d.). https://doc.rust-lang.org/reference/attributes.html

- *Closures: Anonymous Functions that Capture Their Environment - The Rust Programming Language*. (n.d.). https://doc.rust-lang.org/nightly/book/ch13-01-closures.html
- *Advanced Functions and Closures - the Rust Programming Language*. (n.d.). https://doc.rust-lang.org/book/ch19-05-advanced-functions-and-closures.html
- *Using Box to point to data on the Heap - the Rust programming language*. (n.d.). https://doc.rust-lang.org/book/ch15-01-box.html
- *C - Derivable traits - the Rust programming language. (n.d.). https://doc.rust-lang.org/book/appendix-03-derivable-traits.html*

Enhancement 3 Specific:

- *Transaction in mysql - Rust. (n.d.).* https://docs.rs/mysql/latest/mysql/struct.Transaction.html
- *OptsBuilder in mysql - Rust. (n.d.).* https://docs.rs/mysql/latest/mysql/struct.OptsBuilder.html
- *mysql - Rust. (n.d.).* https://docs.rs/mysql/latest/mysql/index.html
- *ConfigBuilder in rocket::config - Rust. (n.d.).* https://api.rocket.rs/v0.4/rocket/config/struct.ConfigBuilder
- *Crates.io: Rust package Registry. (n.d.). crates.io: Rust Package Registry.* https://crates.io/crates/config
- *Drop check - Rust Compiler Development Guide. (n.d.).* https://rustc-dev-guide.rust-lang.org/borrow_check/drop_check.html
- *Drop check. (n.d.).* https://doc.rust-jp.rs/rust-nomicon-ja/dropck.html