Edward Johnson

<u>**Artifact Enhancements #1 – Software Engineering – Enhancement Requirements List:**</u>

- **Enhancement 1:** Implement Rust's error handling using Result<data T, Err> types. Use the the ? operator to manage propagation.
  - *Seen throughout the application, any function that returns -> Result(), also uses the ? operator to handle error propagation.*
- **Enhancement 2:** Move away from using unwrap() statements to avoid panic crashes.
  - *Seen in the application with the use of Option() optional values, the use of ? operator, and the use of Result returns*
- **Enhancement 3:** Implement standard library traits wherever possible to reduce code rewriting, such as with trait Default.
  - *The majority of the standard library traits implemented directly were done so for my custom error types in the errors.rs module. Where I am implementing traits such as std::error::Error for DatabaseError (one of my custom error types).*
- **Enhancement 4:** Implement the Option type for any value that could return None.
  - *Used / implemented throughout my application, anywhere you see the Option keyword, it is noting an option value of Some or None.*
- **Enhancement 5:** Restructure the codebase to utilize composition based design over inheritance.
  - *The codebase was reorganized to utilize structs, implementations of those structs, and there were a few instances of adding trait definitions which are used as interfaces.*
- **Enhancement 6:** Implement proper delegation standards for function calls.
  - *This is implemented in the strict sense, primarily within the database.rs module. My DatabaseManager defines various methods that all database manager should implement, the MySqlDatabase struct then is used to implement the databaseManager trait. This is an example of how in rust we can use a trait implementation to delegate work / operations to the members defined in the initial struct. The pool in this instance is the field/member that is being utilized to perform the actions.*
- **Enhancement 7:** Use enums with match statements for reusable type definitions (in place of constants).
  - P*rimarily used within the errors.rs module, and within the menu.rs module. Used to match error values to their appropriate types, and to match provided user input to corresponding menu options. Match statements are used throughout the application.*

- **Enhancement 8:** Organize the code into distinct modules for better encapsulation.
    - *The original application has been organized into 9 distinct code modules. These modules provide public access functions to allow access to core application functionality when necessary, with the remaining functions being private, and only allowing access from within the module.*
- **Enhancement 9:** Implement structs to define custom data types (instead of classes, etc).
    - *Structs are used throughout the application to represent various objects such as the Client, Employee, MySqlDatabase, Menu.*
- **Enhancement 10:** Utilize traits to provide polymorphism within the application.
    - *Traits definitions were added in two places, with the Identification trait in the firm_models.rs module used to convert client_id values to AVL tree key values, and in the database.rs module with the definition of the DatabaseManager trait that is implemented for the MySqlDatabase struct.*
- **Enhancement 11:** Include relevant annotations, macros, attributes where needed, such as with cloning.
    - *Used on the majority of struct definitions throughout the application, as needed/applicable.*
- **Enhancement 12:** Implement proper memory management techniques for manually defined lifetime definitions.
    - *There were only two significant instances where I needed to work with lifetimes in the application, at least in the current form. The first was in the operation_handlers.rs module, for the Transaction struct and implementation. The second was within the data_structs.rs module, for the find / find value functions of the AVL Tree. These are the only places where I needed to take explicit ownership of data, as it would not make sense to use cloned data in these cases.*

<u>**Artifact Enhancements #2 – Data Structures and Algorithms - Enhancement Requirements List:**</u>

- **Enhancement 13:** Develop an AVL tree data structure for storing and managing Client objects.
    - **Sub Enhancement 1:** Include the following functions, is_empty(), height(), balance_factor(), return_balance(), right_rotate(), left_rotate(), find_value(), insert_value(), remove_node().
        - *Fully functioning AVL Tree implemented/added to the data_structs.rs module.*

- **Sub Enhancement 2:** Use template data types to allow later reusability of the AVL tree.
    - *Functionality added in the AVLTree<T>, and Node<T> structs in the data_structs.rs module. The client_id of a Client object is converted into the key value used in the AVL tree with the Identification trait implemented for the Client object in the firm_models.rs module.*

- **Enhancement 14:** Implement a hash map data structure for storing employee-client associations.
    - *Implemented in the operation_handlers.rs module, in the ClientHandler implementation.*
- **Enhancement 15:** Create an authentication method using secure Rust crates and standards (using argon2).
    - *Implemented using the Authenticator struct of the auth.rs module.*
- **Enhancement 16:** Implement password hashing and comparison functionality for login authorization (using argon2).
    - *Implemented with the hash_password() and authenticate functions of Authenticator implemented struct in auth.rs.*
- **Enhancement 17:** Implement mechanism to handle invalid login attempts (too many / authentication state tracking).
    - *Implemented in the Authenticator struct of auth.rs. Instantiated with values in the constructor function of Authenticator, and tracked / incremented in the authenticate() function of Authenticator.*

<u>Artifact Enhancements #3 – Database - Enhancement Requirements List:</u>

- **Enhancement 18:** Create a database with separate tables for Client and Employee data.
    - *Both tables are accessible via the DatabaseManager and related functions that query the remote database.*
- **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.*

- **Enhancement 20:** Develop a client manager trait to handle client-related operations both locally and in the database.
  - *Implemented with the ClientHandler of operation_handlers.rs*
- **Enhancement 21:** Develop an employee manager trait to handle employee-related operations both locally and in the database.
  - *Implemented with the EmployeeHandler of operation_handlers.rs*
- **Enhancement 22:** Implement functionality to update employee-client pairings both in the database and local storage.
  - *Functionality was added via the following:*
    - *In database.rs module, get_employees() and get_employee() functions 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 the menu.rs module, change_client_employee_pair(), client_pairing_handler(), and get_new_pair_employee_id() functions were added.*
  - *Together these new functions provide an additional main menu option to change the employee that a client is associated with.*
- **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.*
- **Enhancement 24:** Implement methods to retrieve single client objects by client_id (pull from local AVL tree).
  - *This is implemented to allow the client service choices to be updated, uses function named get_client() in the ClientHandler (operation_handlers.rs) Calls the find method of the AVL tree using a client id.*
- **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. The hash map is populated in the ClientHandler constructor function.*

- **Enhancement 26:** Create a method to retrieve all Clients.
  - *It is named get_clients() in the database.rs file. Used to populate the local data structures with client data.*
- **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.*