Team Project: Phase 4

CSE 360 - Group 12
Names: Caitlin Chartier, Prince Goyani, Connor Moon, Keenan Tait, Nishtha Kukreja

Project Overview

The goal of this project is to build a help platform for new and current CSE students. This system should have an easy to navigate interface with content tailorable for each user's knowledge or skill level in various areas, as well as by the level of detail of each piece of information (such as a general overview or summary, or a more detailed piece about a specific topic). This will be made possible through a number of data fields tied to each article, including search tags for each article's difficulty level, the ability to create groupings of articles related to a certain topic, an abstract explaining the information contained within each article, along with the subject the help topic relates to.

Help system data must be able to be sourced from the current semester's Ed Discussion, so that relevant information can be browsed. With this, the materials contained within the system must be able to be quickly replaced or updated, utilizing a number of backup and restore features. This allows handling of scenarios such as when class materials are changed for a new version of a particular class, or when assets referenced in the help materials change such as links to online resources or tools. This is supported by the ability to backup all articles, or backup of only certain groupings of articles, so that data can be handled, stored, and combined in a granular manner, with the ability to revert system data to any previously saved state.

This help platform is integrated with a user login system, where users are able to create accounts of varying types to allow each user a focused, tailored experience, while simultaneously allowing for management of the help system and the data contained within. Specifically, this user role system will include administrators, students, and instructors, who are each given a specialized interface allowing them to access and perform their corresponding actions. For student users, this facilitates the ability for them to find help articles relevant to them in an easy and concise manner. Whereas for admins and instructors, their specialized interfaces makes it straightforward and possible for them to carry out further actions related to their role beyond the simple search and viewing of help articles, such as editing or deleting articles, and handling user accounts (including aiding users in resetting their password). Additionally, admin users are given the ability to backup and restore articles in the system, and admins and instructors are also able to be granted further permissions allowing them the ability to manage certain groups of articles.

The project will consist of 4 phases, each building on what was done in the previous phase.

Phase 1 consists of creating the foundation of the project. In this phase, a GitHub repository is created allowing for secure and asynchronous development, overall project requirements are evaluated, and a working prototype of the user login screen is created including some limited administrator privileges. From here, a foundation is established to allow work to be done in a linear fashion, with further specific features added through the following phases, to work towards completion of the help system in Phase 4. From here, as the project proceeds through to the later phases the requirements, as well as the architecture, design, code, and test cases, are updated to include further project specifications which have been laid out in the various instruction documents for each subsequent phase.

Phase 2 deals with the representation of the data into the software. Specifically, in this phase the interface for the viewing of help items is created, with an emphasis on the user roles of admins and instructors. The interface to view each help item is created, which includes a header displaying a variety of information about the article. This header includes information such as the article's difficulty level, identifiers/tags relating to this article, as well as any restrictions on who is able to view it. In addition to this header, each help item will display its title, a short description/abstract, relevant keywords/phrases, links to reference material, as well as the help article itself. In Phase 2, the implementation is created for methods of searching, addition, removal, updating, backup, and restoration of help articles. System elements created in this phase are fully integrated with system elements created in the first phase, as is the case with each phase of this project.

In Phase 3, the focus of the improvements relate primarily to the student user interface, further implementing the systems and protocols for handling of grouped articles, enhancing the overall security of the database, as well as beginning to incorporate JUnit tests for the code. As this relates to the student user interface, this includes improvements to the ability to search and display articles, as well as the ability to request help from the system operations team when the student is unable to locate particular help information. Also in Phase 3, the security and privacy of information contained within the system is enhanced. Specifically, in the event that memory is dumped and attempted to be analyzed, any sensitive data will not be made visible. In addition to this, protocol was established and implemented to handle correctly displaying or hiding sensitive articles. Only users with the correct permissions will be able to view sensitive article data when searching or attempting to display certain articles. In the case that a user does not have the correct permissions, they are given an alternate non-sensitive version of the data when any sensitive article is attempted to be displayed. Finally, Phase 3 included initially establishing JUnit tests for the code, which will be improved in the final phase.

In Phase 4, the final phase of the project, an overall refinement is carried out over the entire system including its architecture, design, code, internal documentation, and testing suites. It will be ensured that all these aspects of the system are cohesive and functioning properly, and that all design requirements laid out in the project guidelines have been met satisfactorily, including addressing any feedback or changes to the system that have been brought up in previous phases. With this final system refinement and evaluation, user interface elements may be improved, and any wanted additional features added that may make the system more appealing to users, improving on its ease-of-use and which shows a high level of care and detail that went into the development of the system.

User Stories (Requirements)

Because this system includes multiple user account types, there are a number of different user stories that exist to outline the overall requirements of the project.

User Story: First User, Create Administrator Account

“I am the first user and want to create an administrator account so I can launch the platform.”

When a fresh version of this help platform is launched, there will initially be no user accounts. The first account created will be an administrator account.

When the first person attempts to log in, the system assumes that we want to create the first admin account. This requires the input of a desired username and password. This password is prompted to be entered twice to ensure that they match. After entering in this information, the first admin account is created. The system will return to the initial login screen and prompt the user to sign in.

After then performing the first standard login with the newly created account, the user is taken to an interface to finish their account setup. This "Finish setting up your account" page will prompt the user to enter their email address and their name. This is done by entering in data to four different text fields, one each for first name, middle name, and last name, as well as an optional field for a preferred first name to be used within the system when addressing the user.

User Story: Admin, Invite New User To Join Application

“As an admin, I want to give a customized access code so I can invite members.”

Admin accounts are given the ability to invite new users to join the help platform. This can be done using the interface provided after logging in as an admin user role.

The admin will select which role(s) are to be given to the new user to be invited, and after confirming this a one-time code will be generated. This code will allow the new user the ability to create an account from the standard login page.

User Story: Admin, Reset A User Account’s Password

“As an admin, I want to reset a user’s password because they forgot their password.”

In the event that a user requires their password to be reset, an admin must be contacted who can carry this out.

Within the admin login interface, an admin will find a field allowing for the input of a user account name to be reset. After filling in and submitting this information, a one-time password and an expiration date for this password is created. The admin can then provide this information to the user whose account is to be reset.

User Story: Admin, Delete A User Account

“As an admin, I want to delete a user’s account because the user has left the CSE 360 class.”

Within the help system, it is the ability of an administrator to delete a user account if the need arises.

This can be done from within the admin role user interface. Within this interface, the admin will find a field allowing a username to be entered of an account that is to be deleted. Once this username is entered and submitted, a message prompting the admin to confirm the deletion of this account will be shown. The admin will select the “Yes” option if this is correct, and the deletion of the specified user account will then take place.

User Story: Admin, Add Or Remove A User’s Role(s)

“As an admin, I want to change user roles so that I can fix someone having the wrong roles.”

When a certain user accounts role(s) are to be changed, this can be carried out by an administrator.

After logging into the system with the role of admin and proceeding to the admin role interface, a user will find a place to enter the username of an account which is to have role(s) changed. Along with this field requiring their username, the admin is prompted with a selection of which set of roles the account should be updated with. With both of these fields completed, the admin can submit this data which will perform the selected role change on the specified user account.

User Story: Admin, List All User Accounts

“As an admin, I want to list all user accounts so that I can get an overview of everyone who is using the platform.”

An administrator is able to view a list of all user accounts that exist within the system.

An admin user after being logged into the respective role’s interface, will be able to select an option that will display a list of all of the users. This list will include all the user accounts, with each account’s corresponding user name, real name of the user, and the role(s) assigned to that user displayed with a set of codes.

After any changes have been made to this user list such as addition or removal of accounts, or changes made to details of existing accounts, the admin can repeat this process to display a newly updated information set.

User Story: Non-Admin User, Request And Carry Out Password Reset

“As a regular student or administrator, I want to be able to request a password change because my passwords have been compromised.”

When a standard user of the system requires that their password is reset, they can do so by contacting an administrator. After providing their account name to the admin, the admin can generate them a new one-time password. This one-time password has an expiration date that it must be used before.

After receiving this information back from the contacted admin, the user can go through the process of setting up a new account password. The user will use the standard login screen, and after the reset has been performed by the admin, this next login attempt will result in the system requiring that a new password be set. The system will check to verify that this one-time password has not expired before allowing the user to proceed. From here, two text fields are presented, allowing the user to specify a new password, and to confirm that password with the second field. Now, the user is redirected to the login screen where they can perform a standard login with their newly set password. The password reset has been completed, with the new password correctly applied to their account. The one-time code has now been marked as invalid, as it has now already had its one-time use.

User Story: New User, Sign Up With One-Time Code

“As a person who has just been invited to the platform, I want to use my access code so that I can join.”

A new user may be invited to sign up and create an account on the help platform after being provided with a one-time code from an administrator.

After having received this one-time code from an admin, the new user can access the standard login screen. From here, instead of entering in their username and

password as an existing user would, they will look for the input field accepting an invitation code.

The provided one-time code can be entered here, which will then bring the user to an interface prompting them for a desired username and password. This password must be entered into two fields, which ensures that the password is matching. At this point the new account is created automatically with the role(s) specified by the administrator upon generation of the code, and the system redirects the user back to the login screen.

Now that the account is created, the setup of the account can be finished. After logging in with the newly created login details, the user is brought to an interface to finish their account setup. This interface includes four text fields, one each for first name, middle name, and last name, as well as an optional field for a preferred first name. After submitting this information, the user continues with their login. Multi-role users are then prompted to select their desired role for the current session, whereas role selection is done automatically for single-role users. The home page for the corresponding account role is then displayed, and the sign up and login process has been completed.

User Story: Multi-Role User, Select Role Upon Sign-In

“As an instructor, I want to be able to select If I want to browse as a student or instructor so that I can experience what my students will see.”

Because this system supports multiple account roles to be assigned to each user account, a selection of which role for the current session must be possible when logging in. This selection is offered to multi-role users upon logging into the system.

When attempting to log in with the standard login interface, after a multi-role user has entered their login information successfully, they are prompted with the question of which account role to select for this session. This will allow them to select between the roles available to them. After this selection is made, the sign in process may continue, bringing the user to their requested role’s corresponding interface.

For users with only a single role, this prompt is skipped and they are simply taken to their corresponding role’s homepage.

User Story: Student Or Instructor, Successful Login

“As a student or instructor, I want to be brought to my home page so that I can start with relevant information.”

When a user successfully logs in with the role of student or instructor, they are brought to a home page corresponding to that role. For Phase One, upon reaching this homepage, the user is presented with only one option: to log out.

User Story: All Users, Log Out

“As anyone using the platform, I want to be able to log out so that other people can use their accounts on the same machine.”

All user role interfaces, regardless of their type, will feature a log out button on their interface homepage.

When a user has been already logged into the system, and they wish to now log out of the system, they can proceed to their interface homepage. From here, the user can locate the “Log Out” button, which once selected, will log the user out of the system, and navigate them back to the standard login system.

User Story: Admins or Instructors, Place Articles Into Groups

“As an admin or instructor, I want to place articles into groups in order to easily separate different semesters”

When creating or updating articles, there will be a section where groups can be specified. These groups can be used elsewhere to easily organize and sort through the articles on the database.

User Story: Admins or Instructors, Backup/Restore Articles

“As an admin or instructor, I want to be able to backup articles to an external device so that I can save this semester’s articles in preparation for next semester.”

Admins and instructors need additional functionality to backup the database to an external device, and the option to later restore them. This backup can be configured to only backup certain article groups. Additionally, when restoring they will have the option to completely clear the database and replace it with the backup or merge the files. This will look at each article’s unique id, and if a conflict is found the article will not be merged.

User Story: Admins or Instructors, Backup/Restore Articles

“As an admin or instructor, I want to be able to interact with any of the help articles so that I may view, create, update and/or delete them”

Admins and instructors need to be able to interact with the help articles in order to effectively use the database. Both will have the ability to see, change, create, and delete articles on the database. Additionally, they will be able to list all articles in a group or subset of groups, which can help them find specific article(s).

User Story: Admins, Remove Admin Rights For Entire System, Or A Group

"As an admin, I may need to be able to remove admin rights from other users of the help system as a whole, or for specific article groups."

One of the roles of admins in the help system is the ability to manage users of the system, but also to manage groups that are contained within the system, namely article groups. From the admin interface, the action can be selected to remove admin rights from a particular group. This will be carried out, except in the case where that change would mean that there is no remaining user with admin rights for that particular article group. In that case, this action will not be carried out.

User Story: Admins, List Users Given Admin Rights For A Special Access Group

"As an admin, I want to be able to view a list of users given admin rights for a specific special access group."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access group. The system will then provide this admin user with a list of all users which have been given admin rights for this particular special access group.

User Story: Admins, Manage Users Of A Special Access Or General Group

"As an admin, I want to manage the users of a special access or general group."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access or general group. The system will then provide this admin user with the option to either view all users, add users, or remove users from this specific group. These options to add or remove users, allows the admin to add or remove students, instructors, and admins from that particular group.

User Story: Admins, For A Special Access Group, List Sensitive Articles

"As an admin, for a specific special access group, I want to be able to view a list of the articles within this group where decrypted access is limited."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access group and select to list the sensitive articles in this group. The system will then provide this admin user with a list of all articles in this group where decrypted access is limited.

User Story: Admins, First Instructor Added To A Special Access Group Is Given Admin Rights

"As an admin, when adding the first instructor to a special access group, they must be given decrypted viewing access and admin rights in that group by default."

From within an admin's user interface, when they are attempting to add the first instructor to a special access group, they can carry this out by selecting the relevant option the same way they would any time when adding an instructor to a special access group. However, if this is the first instructor to be added, the system will by default grant this instructor both the rights to view decrypted articles and admin rights in this special access group. This ensures that there is at least one user in that particular group that has the full rights to manage it.

User Story: Admins, For A Special Access Group, List Instructors With Admin Rights

"As an admin, for a specific special access group, I want to be able to view a list of the instructor users in this group who have been given admin rights."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access group and select to list the users group who have been given admin rights. The system will then provide this admin user with a list of all instructors in this group with this privilege level.

User Story: Admins, For A Special Access Group, List Instructors With Decrypted Viewing Rights

"As an admin, for a specific special access group, I want to be able to view a list of the instructor users in this group who have been decrypted viewing access."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access group and select to list the instructor users group who have been given decrypted viewing access. The system will then provide this admin user with a list of all instructors in this group with this privilege level.

User Story: Admins, For A Special Access Group, List Students With Decrypted Viewing Rights

"As an admin, for a specific special access group, I want to be able to view a list of the instructor users in this group who have been decrypted viewing access."

From within an admin's user interface, they have available to them the ability to enter in the name of a special access group and select to list the student users group who have been given decrypted viewing access. The system will then provide this admin user with a list of all students in this group with this privilege level.

User Story: Students, Send Message To Help System

“As a student, there might arise a need to be able to send a message such as a help request to the administrators of the help system. This message will vary according to the current need.”

From within a students user interface, they have available to them the ability to compose and send a message to the admins of the help system which can act as a help request. The student is able to select whether they would like to send a generic message, or a specific message that they can edit. A list of generic messages are provided which the student may select. Alternatively, the student may select to enter in their own message, where they are able to write their own custom message to be sent as a help request.

User Story: Student And Instructor, Perform Article Search

“I want to search through the article database and return a list of articles which are suited to me, based on the varying search criteria available.”

From within a user’s interface, they are given an option to perform a search to display a list of articles which fit a set of search criteria. The user may opt to select a content difficulty level, including beginner, intermediate, advanced, expert, or all. Along with this, the user is given the ability to enter in a specific criteria for character strings contained within the title, the author, or the article’s abstract, as well as the ability to search for a specific article grouping identifier. The system will then display a list of articles which match this student’s specific search request. This search result will include a top-line which includes the group name if one has been specified, and the number of articles that match each level, followed by the list of matching articles displayed in a short form. From here, the user can choose to perform another search or to change to a different screen.

User Story: Admin And Instructor, Add A Help Article

“As an admin or instructor, I want to create a new article.”

From within an admin or instructor’s interface, they are given the ability to carry out special actions not given to standard student users, including adding a new article. An admin or instructor may select the option to add a new article, where they are prompted for the article data such as the title, author, body text, etc, and then this can be submitted, where it is then added to the help system.

User Story: Admin And Instructor, Manage A Help Article

“As an admin or instructor, I want to manage a specific help article, including editing or deleting that particular article.”

From within an admin or instructor's interface, they are given the ability to carry out special actions not given to standard student users, including managing existing articles. An admin or instructor may select the option to manage an existing article, where they are prompted if they would like to delete the article, or edit a particular data field for that article. After receiving the user's choice, the help system will then carry out this change.

User Story: Instructor, Manage Groups

"As an instructor, I want to manage groups, including creation, viewing, editing, and deleting these groups."

From within an instructor's interface, they are given an option to manage a special access or normal group. From here, they may select whether they would like to create a new group, or view or manage an existing group. If they choose the latter, they are prompted to enter in a group name and from here, they can select whether to add, remove, or change a specific user's permissions in the specific group entered.

User Story: Instructor, View Help System Users

"As an instructor, I want to view the list of student users who are allowed access to the help system."

From within an instructor's interface, they are given an option to view the list of current students given access to the help system. After selecting this option, the instructor is given a list of students who are currently allowed access to the system.

User Story: Instructor, Manage Help System Users

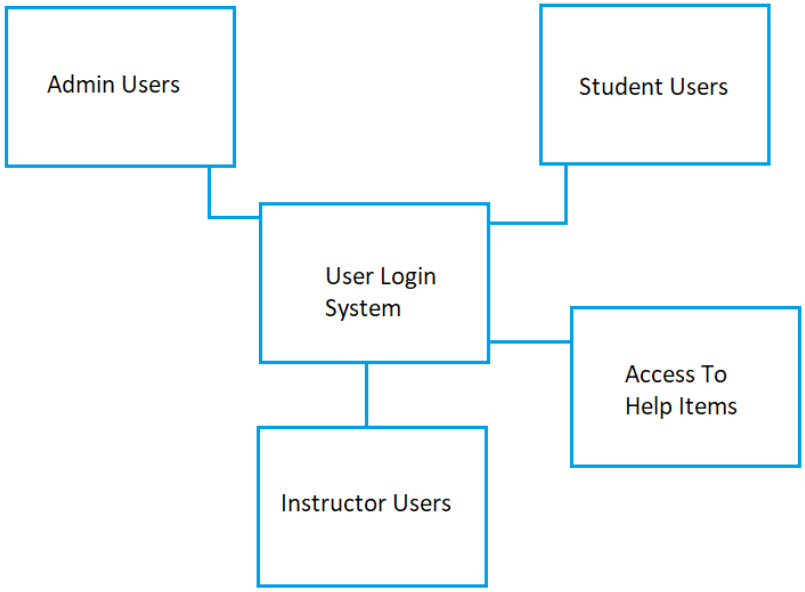
"As an instructor, I want to manage student users who are allowed access to the help system."

From within an instructor's interface, they are given an option to manage students who are currently given access to the help system. The instructor may select an option to add a new user, or they may select an option to delete a student from the help system. In the latter case, the instructor is prompted for a student's name to be entered, which after submitting this, will have their student user account removed from the system.

Architecture

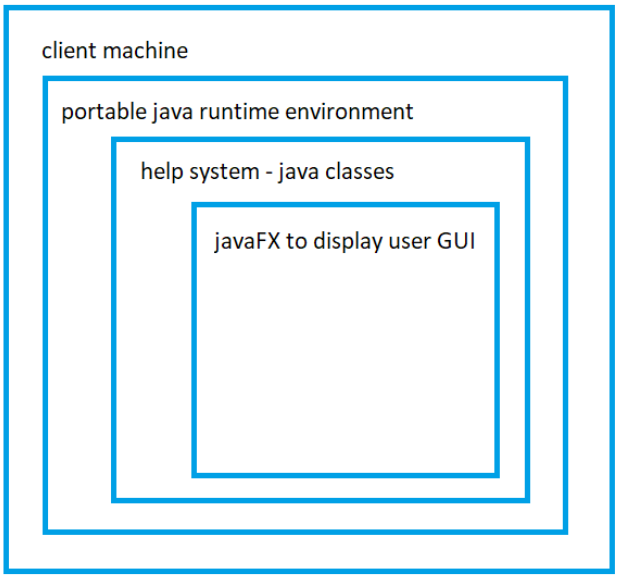
Context Diagram

In the context diagram shown below, the system is used by multiple different types of users. Each of these types of users goes through the system's user login system, which grants each type of user appropriate access to the help items contained within the system. Whether the particular user is given certain privileges, such as the ability to add and remove help items, is based on what type of user role is associated with their account, and their currently chosen login role.



Architectural Diagram

The major reused components of the help system's architecture can be seen below in the architectural diagram. The help system uses a combination of major components to allow the system to function for all users.

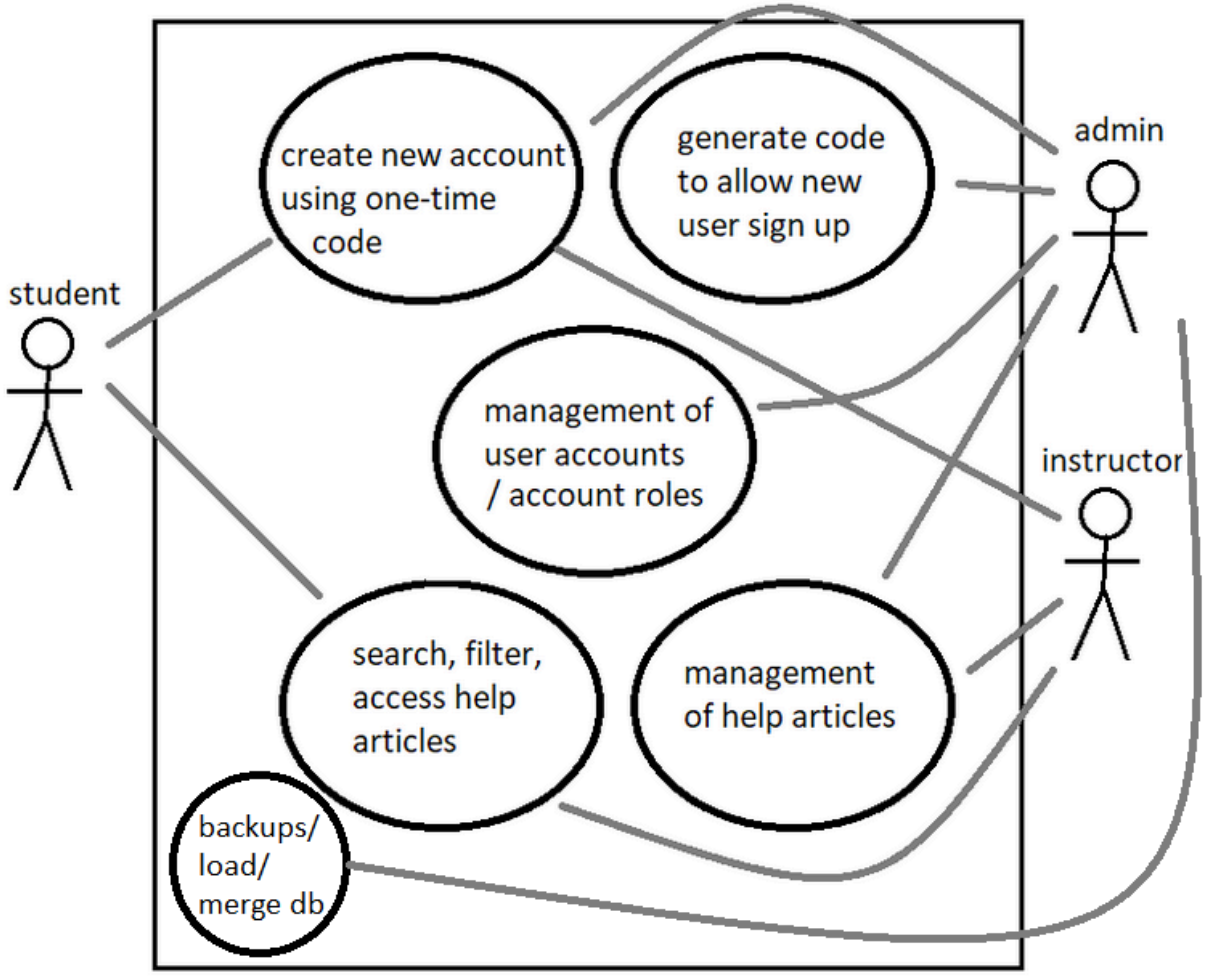


Specifically, in the architectural diagram, there is a hierarchy of different items to allow a user to use the help system. In order for a user to access the help system, they will begin by using their own computer. This computer must have the Java Runtime Environment installed, which will allow that computer to run Java programs, such as the help system. From here, the program for the help system which is coded using a number of Java classes is able to run, which does the work for the user login system as well as the management of help articles contained within the system. Additionally, the JavaFX library is then used together with the backend classes performing the management of system data, to present a nicely packaged GUI for the user to interface with the help system. The combination of these different layers of architecture allows for the help system to be utilized by different users, but each in a simple, easy-to-use manner.

Design

Use Case Diagram

The UML use case diagram, consisting of a graphical representation of the interaction between actors, who are the users of the system, each use case, and the system itself, is shown below. In this diagram, the square represents the boundary of the system, where things within the box are an integral part of the system. Specific use cases are shown within the circles.



Class Responsibility Collaborator (CRC)

Class Responsibility Collaborator (CRC) card diagrams, which are used in agile development to show each class, the responsibilities of the class, and the collaborators, which are other classes that interact with this primary class, are shown below.

In this first CRC card diagram, the class DatabaseHelper is shown. This class is what manages the backend of the help system, which includes the user account database. This class is responsible for allowing the main App class to access data related to the user accounts, such as displaying the list of users, handling logins and connections to the database, and updating user passwords when necessary.

Class Name: DatabaseHelper	
Responsibilities: Mange user connections to database. Displaying list of users. Updating user roles. Check if database is empty. Handle user logins. Update user passwords.	Collaborators: App class

The ArticleDatabase CRC card shown below describes the responsibilities and class relations of the ArticleDatabase. This class provides structured calls that can be used by the user interfaces within the help system, so that the DatabaseHelper can be called with the correct data/arguments to properly carry out database functions.

Class Name: ArticleDatabase	
Responsibilities: Gives the user an interface to manage the database. Provides structured calls to DatabaseHelper.	Collaborators: DatabaseHelper

The CRC diagram shown below is for the App class. This class handles the front end of the help system, and is responsible for providing the correct user interface elements to each user depending on their current login role.

Class Name: App	
Responsibilities: Provide the user an interface for each part of the system. Handle user input requests. Act as an interface between the user and DatabaseHelper. Generate one-time codes.	Collaborators: DatabaseHelper

The CRC card for the LoginPage class is shown below. This is the class that handles all of the GUI, including the login page. Many of the functions relating to user information and flow are done by collaborating with the App class which contains the user account information. ArticleDatabase is needed to access the database, which stores the help articles and is needed to backup and restore the database to/from external storage.

Class name: LoginPage	
Responsibilities: Handles all of the GUI and managing where to direct the user	Collaborators: App, ArticleDatabase

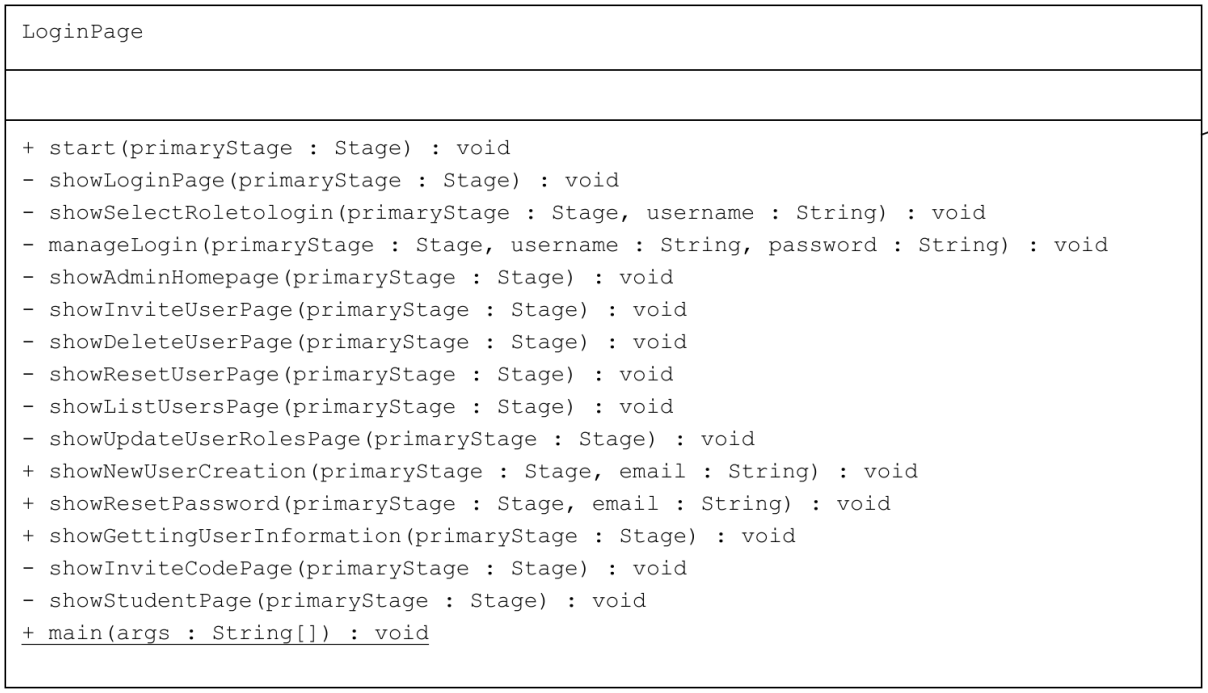
UML Class Diagrams

Shown below are UML class diagrams for each of the classes that make up the help system. For each of these class diagrams, the name of the class is listed, along with the attributes/variables that are a part of this class, as well as the operations/methods that the class has available.

The first picture is all of the classes together, and afterwards is each of them individually.


<<utility>> ArticleDatabase
- scanner : Scanner {readOnly}
+ databaseHelper : DatabaseHelper
+ main(args : String[]) : void
+ callMenu(menuOption : String) : void
+ defaultFlow() : void
+ callDisplayArticles() : void
+ callDisplayArticleByKey(input : String) : void
+ callDisplayArticleByGroups(searchGroupings : String) : void
+ callDisplayArticleByKeywords(searchKeywords : String) : void
+ callDeleteArticleByKey(input : String) : void
+ callCreateArticle(newTitle : String, newBody : String, newAuthor : String, new
+ callCreateArticle(newTitle : String, newBody : String, newAuthor : String, new
+ callEditArticle(key : String, editTitle : String, editBody : String, editAutho
+ callEditArticle(key : String, editTitle : String, editBody : String, editAutho
+ callBackupFile(backupFilename : String) : void
+ callBackupByGrouping(backupFilename : String, groupingIDs : String) : void
+ callLoadFile(loadFilename : String) : void
+ callLoadFileMerge(loadFilename : String) : void


DatabaseHelper
- encryptionHelper : EncryptionHelper
- statement : Statement
- connection : Connection
~ PASS : String {readOnly}
~ USER : String {readOnly}
~ DB_BACKUP_ROOT : String {readOnly}
~ DB_ROOT : String {readOnly}
~ DB_URL : String {readOnly}
~ DB_NAME : String {readOnly}
~ JDBC_DRIVER : String {readOnly}
+ DatabaseHelper()
+ connectToDatabase() : void
+ connectToDatabase(loadFileURL : String) : void
- createTables() : void
+ backupToFile(backupFilename : String) : void
+ backupByGrouping(backupFilename : String, groupingIDs : String) : void
+ saveDatabase() : void
+ loadFromFile(loadDb : String) : void
- extractBackup(zipFilePath : String) : String
+ loadFromFileMerge(loadDb : String) : void
+ isDatabaseEmpty() : boolean
+ createArticle(title : String, body : String, author : String, abstrac : String, keywords : String, references : String, c
+ createMergedArticle(title : String, body : String, author : String, abstrac : String, keywords : String, references : Str
+ createArticle(title : String, body : String, author : String, abstrac : String, keywords : String, references : String, c
+ displayArticles() : void
+ displayArticleByKey(key : int) : void
+ displayArticlesByGrouping(grouping : String) : void
+ displayArticlesByKeywords(grouping : String) : void
+ checkArticleByKey(key : int) : int
+ checkSensByKey(key : int) : int
+ deleteArticleByKey(key : int) : void
+ editArticleByKey(key : int, newTitle : String, newBody : String, newAuthor : String, newAbstrac : String, newKeywords : S
+ editArticleByKeySens(key : int, newTitle : String, newBody : String, newAuthor : String, newAbstrac : String, newKeywords
+ closeConnection() : void

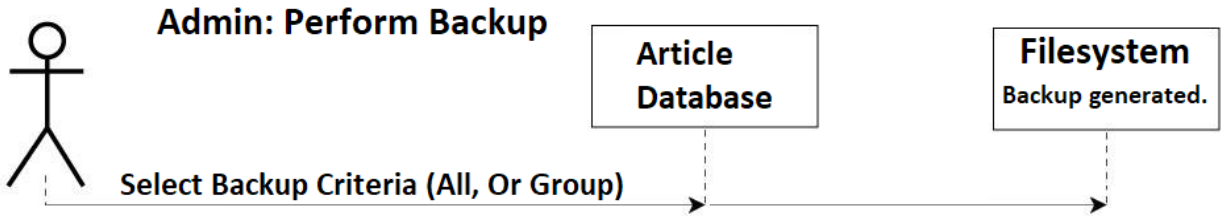

LoginPage
+ start(primaryStage : Stage) : void
- showLoginPage(primaryStage : Stage) : void
- showSelectRoletologin(primaryStage : Stage, username : String) : void
- manageLogin(primaryStage : Stage, username : String, password : String) : void
- showAdminHomepage(primaryStage : Stage) : void
- showInviteUserPage(primaryStage : Stage) : void
- showDeleteUserPage(primaryStage : Stage) : void
- showResetUserPage(primaryStage : Stage) : void
- showListUsersPage(primaryStage : Stage) : void
- showUpdateUserRolesPage(primaryStage : Stage) : void
+ showNewUserCreation(primaryStage : Stage, email : String) : void
+ showResetPassword(primaryStage : Stage, email : String) : void
+ showGettingUserInformation(primaryStage : Stage) : void
- showInviteCodePage(primaryStage : Stage) : void
- showStudentPage(primaryStage : Stage) : void
+ main(args : String[]) : void


<<utility>> App
- datas : String[]
- activeRole : String
- password : String
- username : String
- scanner : Scanner {readOnly}
- databaseHelper : DatabaseHelper {readOnly}
+ getActiveRole() : String
+ setActiveRole(role : String) : void
+ main(args : String[]) : void
+ connect() : boolean
+ fristUser() : boolean
+ createUser(testEmail : String, testRole : String) : boolean
+ userResetEmail(username : String) : boolean
+ resetPasword(email : String, passWord : String) : boolean
+ logout() : void
+ setupAdministrator(userName : String, passWord : String) : boolean
+ setupStudent(userName : String, passWord : String, email : String) : boolean
+ appLogin(loginName : String, credentials : String) : String
+ loginInvitedUser(email : String, inputCode : String) : String
+ setupUserInformation(fristName : String, middleName : String, lastName : String) : boolean
- generateRandomOneTimeCode() : char[]
- verifyEmail(email : String) : boolean
+ deleteUser(inputUser : String) : boolean
+ addOrRemoveRole(inputUser : String, whatAction : String, inputRole : String) : boolean
+ listOfUser() : String[][]
+ adminHome() : void
+ studentHome() : void
+ instructorHome() : void
+ parentsHome() : void

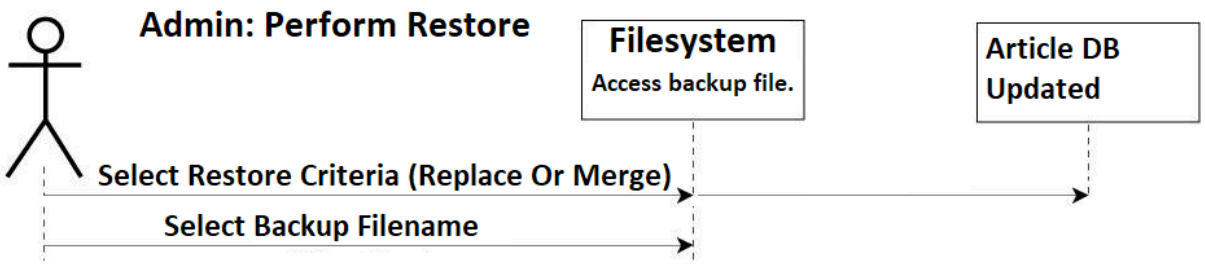
Additional Design Diagrams

For Phase 3, additional design diagrams have been added to illustrate some of the interactions between the user and the system.

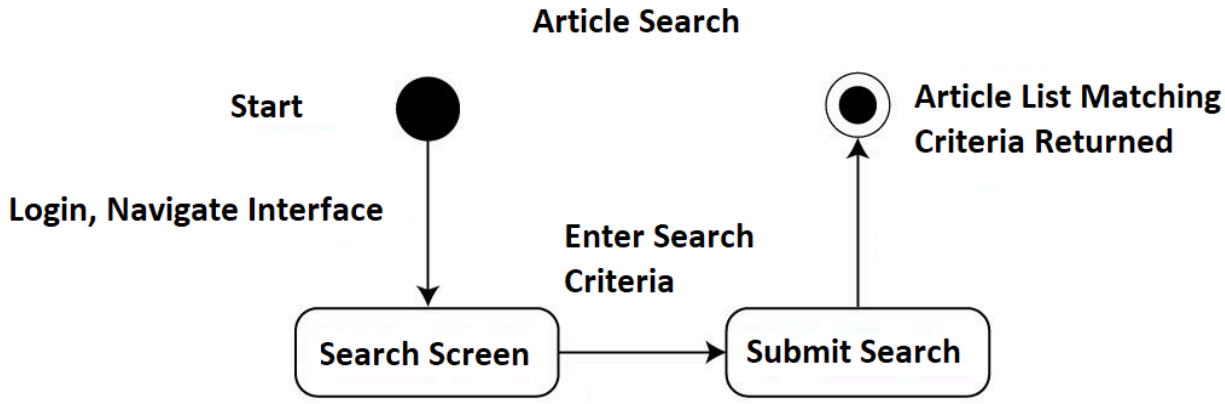
The above sequence diagram shows the general form that an article list search request takes between the user and the article database.



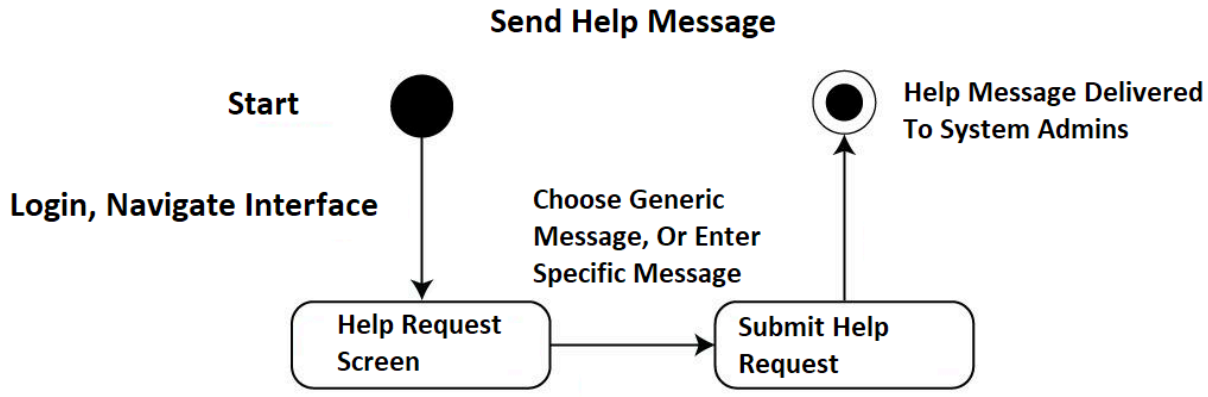
The above sequence diagram illustrates the interaction that occurs between the user and an admin user when attempting to generate a new backup.



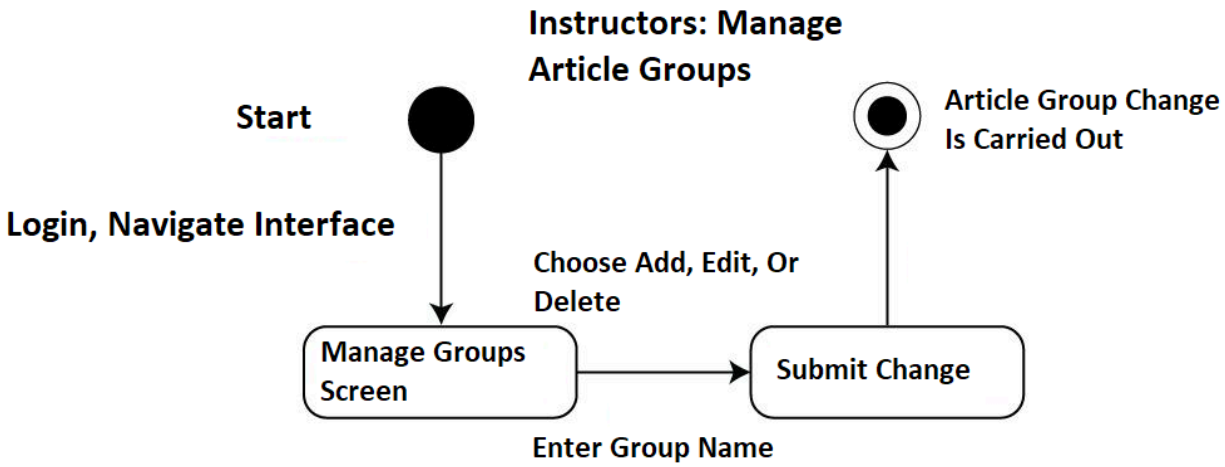
The above sequence diagram shows the steps in the user-system interaction when an admin attempts to perform a restore to load a backup file into the database.



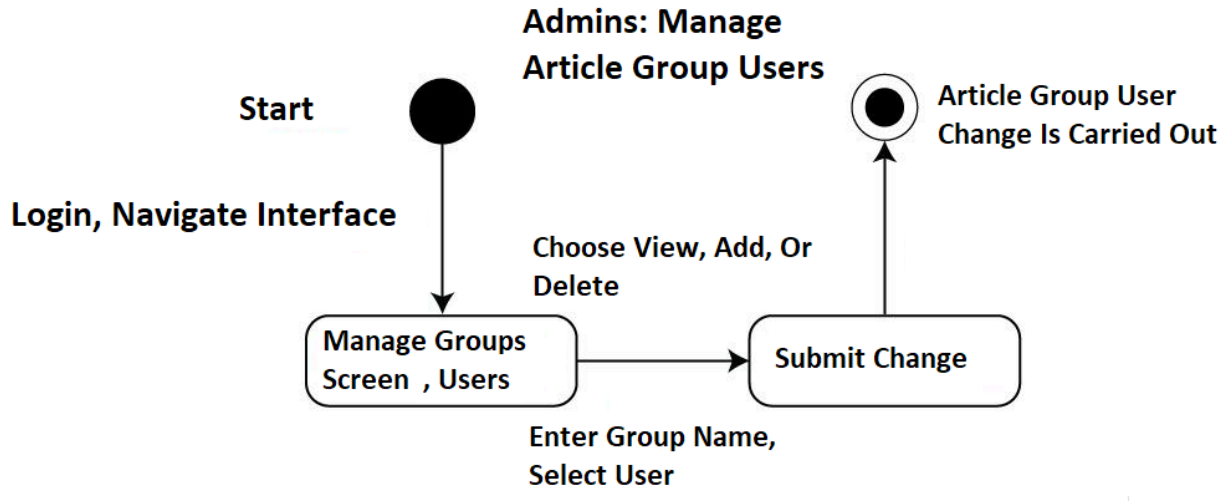
The above state diagram shows how an article search is carried out, from first opening the help system, to having a list of matching articles displayed.



The above state diagram shows how a help message is composed and submitted to the system admins.



The above state diagram shows how an instructor would manage article groups, this includes special access and general access groups, and includes creation, editing, or deletion of these groups.



The above state diagram illustrates how an admin would manage users of a specific special access or general access group.

Testing Requirements

For this help system project, there were many requirements that were given throughout the different phases of the project. For the final phase, these specific given requirements were combined together and sorted into an ordered list where they could each be addressed and tested appropriately whether using a JUnit test or a manual test.

These requirements were grouped together by their initial phase within the project, which worked to neatly organize them in a way that makes sense, with subsequent requirements in the list building on the requirements satisfied from earlier items in the list. This makes sense because this is the same order that the requirements were given during the help system development throughout the phases, and also therefore, the same order that the requirements were able to be implemented and integrated together. This complete requirements list which outlines the test cases is as follows:

Phase One Requirements

1. The very first person to log into the application (e.g., the list of users is empty) is assumed to be an Admin. The system requires the first user to specify a username and password. An account is created for that username and password, and that user is assigned the role of Admin. At that point, the user is directed back to the original login.
2. When creating a password, the password must be entered two times, and they must match.
3. Before a user can use the system, when logging in again they are first taken to a "Finish setting up your account" page. The "Finish setting up your account" page requires the user to specify an email address and their name. There are four fields associated with a name: first, middle, last, and optionally preferred first name. (If the user specifies an optional preferred first name, this name will be used when displaying messages to that user from the application.)
4. The system must support multiple roles. The following are required minimum roles: Admin, Student, Instructor.
5. A user may have more than one role. If a user has more than one role, after signing in, the user must specify which role is appropriate for this session. If the user has just one role, the user is taken to a page to the home page for that role. For Phase 1, the Student and Instructor role home pages have only one option, and that is to log out.
6. An Admin can perform the following: Invite an individual to join the application. A one-time code is provided that allows a new user to create an account. The standard login page allows the user to provide a username to start the login process or a different input field in which they can enter the invitation code. The Admin must specify which role(s) this invited user is being given when producing the invitation.
7. An Admin can perform the following: Reset a user account. A one-time password and an expiration date and time is set. The next time the user tries to log in, they must use the one-time password, and the only action possible is to set up a new password. Before being given access to set up a new password, the system checks to see if the

date and time are proper given the deadline. Once the new password has been set, the user is directed back to the login page. Logging in with the one-time password resets the flag so it can't be used again.

8. An Admin can perform the following: Delete a user account. An "Are you sure?" message must be answered with "Yes" to do the delete.
9. An Admin can perform the following: List the user accounts. A list of all the user accounts with the user name, the individual's name, and a set of codes for the roles is displayed.
10. An Admin can perform the following: Add or remove a role from a user.
11. An Admin can perform the following: Log out.
12. All other users can perform the following: At the login page, fill in the one-time invitation code to be allowed to establish an account. The only action allowed when establishing an account is to specify a username and password. An account is created for that username and password with the role(s) associated with the one-time invitation.
13. At that point, the user is directed back to the original login. As described above, they must finish setting up their account. Once the account is set up, they then have access to the home page to which they have been assigned or to the page where they can select which role is appropriate for this session and then the home page for that role.

Phase Two Requirements

14. Each article is given a unique long integer identifier when created, so duplicates can be easily detected.
15. The admin and the instruction team roles are enhanced by providing commands to back up and restore the help system data to an admin/instructor-named external data file.
16. An option is provided to remove all the existing help articles or merge the backed-up copies with the current help articles. When the unique long integer identifiers match, the backed-up copy will not be added.
17. The system shall support a mechanism to support multiple groups of help articles. Grouped articles can be backed up, so only the articles in the group are in the backup. For example, articles for a CSE360 implementation might have one group for Eclipse articles and another for IntelliJ articles. This allows instructors or admins to set up the help system for two different help systems based on which IDE is used in the course. Similarly, there may be a group for H2 database articles and another for SQL Fiddle articles. It is also possible for a single article to belong to more than one group (e.g., Eclipse and H2).
18. Both admins and instructors may create, update, view, and delete help articles.
19. Both admins and instructors may list all the help articles and subsets of the help articles in a group or multiple groups.

Phase Three Requirements

20. No role has the right to remove admin rights if that means no one will have admin rights for the whole help system, a general group, or a special access group.

21. The system supports special access groups consisting of: A list of articles in the group where the body of the article is encrypted and decrypted access is limited. (E.g., A collection of help articles that contains proprietary, private, or other kinds of sensitive information.
22. The system supports special access groups consisting of: A list of admins given admin rights to create, read, update, and delete access rights to this group. (Admins **do not** automatically have admin rights to special access groups and do not have the right to view the bodies of articles in this group.)
23. The system supports special access groups consisting of: The first instructor added to a special access group is given the right to view the bodies of articles in the group and admin rights for this group.
24. The system supports special access groups consisting of: The default rights for new instructors added to this group do not include admin rights for this group.
25. The system supports special access groups consisting of: A list of instructors who have been given rights to view the decrypted bodies of articles in this group.
26. The system supports special access groups consisting of: A list of instructors given admin rights for this group.
27. The system supports special access groups consisting of: A list of students given viewing rights to the decrypted bodies of articles in the group.
28. The student role is enhanced with additional commands: Students may always perform a set of actions, including quitting the application, sending a generic message to the help system, and sending a specific message to the help system.
29. The student role is enhanced with additional commands: Specific messages are those where the student cannot find the help information they need. In this last case, the student specifies what they need and cannot find. The system adds a list of the search requests the student has made.
30. The student and instructor role is enhanced with additional commands: Set the content level of the articles to be returned (e.g., beginner, intermediate, advanced, expert).
31. The student and instructor role is enhanced with additional commands: Specify a group of articles to limit returned articles to just those items in a group (e.g., a group may be help articles for a particular assignment).
32. The student and instructor role is enhanced with additional commands: Searching uses words, names, or phrases in the title, author, or abstract.
33. The student and instructor role is enhanced with additional commands: When a search is performed, the system will display the following.
    - The first line displayed specifies the group that is currently active,
    - The number of articles that match each level,
    - A list of articles matching the search criteria in a short form. The short form includes a sequence number (1 - n), the title, the author(s), and the abstract.
34. The instructor role is enhanced with additional commands: Instructors may specify a group or a special access group to reduce the length of returned items.
35. The instructor role is enhanced with additional commands: Instructors may view the details of a help article from the returned list of items.

36. The instructor role is enhanced with additional commands: Instructors may create, view, edit, and delete help articles.
37. Instructors may create, view, edit, and delete general article groups. (See above about instructors' rights concerning general groups.)
38. Instructors may create, view, edit, and delete special access article groups. (See above about instructor's rights concerning special access groups.)
39. Instructors may backup and restore articles and groups of articles.
40. Instructors may add, view, and delete students from the help system and general groups.
41. Instructors with special group rights for a group may add, view, and delete students from that group.
42. Admins may create, delete, back up, and restore help articles. (They do not have the right to view or edit the body of any help system article.)
43. Admins may be given admin rights over general article groups. (See above about admin rights concerning general article groups.)
44. Admins may be given admin rights over special access article groups. (See above about admin rights concerning special access article groups.)
45. Admins may backup and restore articles, groups of articles, and special access groups of articles.
46. The admin role is enhanced with additional commands: Admin may add, view, and remove students, instructors, and admins from general groups. (See the limitation above that requires at least one admin to have admin rights to every group or special access group.)

After creating this requirements list to lay out the various test cases, either JUnit or manual tests were created. The test to go with each of the above numbered help system requirements can be found below.

Tests

Requirement 5: Users with Multiple Roles Must Select a Role
Test Type: Manual Test
-This functionality is part of the user interface, so manual testing is better suited for it. The test requires interacting with a dropdown menu to select a role.
Setup:
-Add a user with multiple roles (e.g., Admin and Instructor) to the database.
-Log in with that user's credentials.
Inputs:
-Username: multiRoleUser
-Password: password123
-Select the role: Instructor
Outputs:
-The system should direct the user to the Instructor Homepage.
-App.getActiveRole() should update to instructor.
Assessment:
-This test passes if the Instructor Homepage is displayed and the role is correctly set in the backend. It fails if the wrong homepage is shown or the role is not updated.
Coverage:
-This test covers the primary functionality. Additional edge case tests could check what happens -if no role is selected or the user has only one role.

Requirement 6

An Admin can perform the following: Invite an individual to join the application. A one-time code is provided that allows a new user to create an account. The standard login page allows the user to provide a username to start the login process or a different input field in which they can enter the invitation code. The Admin must specify which role(s) this invited user is being given when producing the invitation.

Requirement 1: Invite Users
- Test Type: Manual Test
- Setup:
1. Login as Admin.
2. Navigate to the "Invite User" section in the Admin Dashboard.
- Inputs:
- Email: `testuser@example.com`
- Role: `Student`
- Outputs:
- Email confirmation sent to the user.
- Invite entry appears in the system's pending invites.
- Assessment:
- Verify the email was received with the correct invite code.

- Validate that the user appears in the invite list.
- Coverage:
- Tests invite functionality with valid inputs. Additional test cases should cover invalid email formats and duplicate invites.

Requirement 7
An Admin can perform the following: Reset a user account. A one-time password and an expiration date and time is set. The next time the user tries to log in, they must use the one-time password, and the only action possible is to set up a new password. Before
being given access to set up a new password, the system checks to see if the date and time are proper given the deadline. Once the new password has been set, the user is directed back to the login page. Logging in with the one-time password resets the flag so it can't be used again.

Requirement 2: Reset User Accounts
- Test Type: JUnit Test
- Setup:
- Create a test user in the database using the `createUser` method.
- Inputs:
- Username: `testuser`
- Outputs:
- One-time password is generated.
- Password expiration date is set.
- Assessment:
- Confirm the one-time password and expiration details in the database.
- Validate that the user is prompted to reset the password on their next login.
- Coverage:
- Single test case for valid password reset.
- Additional test cases for expired passwords or invalid usernames.

Requirement 8

An Admin can perform the following: Delete a user account. An "Are you sure?" message must be answered with "Yes" to do the delete.

Requirement 3: Add or Remove Roles
- Test Type: JUnit Test
- Setup:
- Create a test user with a basic role.
- Inputs:
- Add Role: `Instructor`
- Remove Role: `Student`
- Outputs:
- Updated role list for the user.
- Assessment:

- Verify that the roles are updated correctly in the database.
- Confirm that removed roles no longer appear.
- Coverage:
- Test with both adding and removing roles.
- Include invalid role names and attempts to remove non-existent roles.

Requirement 9 - 10

An Admin can perform the following: List the user accounts. A list of all the user accounts with the user name, the individual's name, and a set of codes for the roles is displayed.
An Admin can perform the following: Add or remove a role from a user.

Requirement 4: Add Special Access Groups
- Test Type: Manual Test
- Setup:
1. Login as Admin.
2. Navigate to the "Manage Groups" section.
- Inputs:
- Group Name: `TestGroup`
- First User: `Instructor` with Admin rights.
- Outputs:
- Group appears in the group list.
- Admin rights are assigned to the first user.
- Assessment:
- Verify group details in the database.
- Confirm that subsequent users are added with default rights.
- Coverage:
- Valid group creation and admin assignment.
- Additional cases for duplicate groups and invalid group names.

Requirement 11 - 13

11. An Admin can perform the following: Add or remove a role from a user.
12. An Admin can perform the following: Log out.
13. All other users can perform the following: At the login page, fill in the one-time invitation code to be allowed to establish an account. The only action allowed when establishing an account is to specify a username and password. An account is created

Requirement 5: Search Articles
- Test Type: JUnit Test
- Setup:
- Populate the database with sample articles.
- Inputs:
- Query: `Sample`

- Level: `Intermediate`
- Group: `TestGroup`
- Outputs:
- A list of matching articles.
- Assessment:
- Validate the correctness of the returned articles.
- Check that non-matching articles are excluded.
- Coverage:
- Single test for valid search inputs.
- Include edge cases such as empty queries and invalid groups.

Implementation

JUnit Tests

```
package com.educationCenter.Articler_Database_Handler;

import com.educationCenter.App;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class ArticleDatabaseTests {

    @BeforeAll
    void setup() {
        // Initialize the database connection
        ArticleDatabase.connect_dataBase();
    }

    @AfterAll
    void teardown() {
        // Cleanup and close the database connection
        if (ArticleDatabase.articleDatabaseHelper != null) {
            ArticleDatabase.articleDatabaseHelper.articleCloseConnection();
        }
    }

    @Test
    void testAddGroupToSpecialAccess() {
        String groupName = "TestGroup";
        boolean isGroupAdded = ArticleDatabase.addGroupToSpecialAccess(groupName);
        assertTrue(isGroupAdded, "Group should be added successfully.");
```

```
        // Verify the group exists
        String[] groups = ArticleDatabase.returnGroupsFromUser(App.getUsername());
        assertNotNull(groups, "Group list should not be null.");
        assertTrue(groups.length > 0, "Group list should contain at least one group.");
        assertTrue(groups[0].equals(groupName), "The group name should match the added
group.");
    }

    @Test
    void testRemoveGroupFromSpecialAccess() {
        String groupName = "TestGroup";
        boolean isGroupRemoved =
ArticleDatabase.removeGroupFromSpecialAccess(groupName);
        assertTrue(isGroupRemoved, "Group should be removed successfully.");
    }

    @Test
    void testReturnUsersFromGroup() {
        String groupName = "SampleGroup";
        String[] users = ArticleDatabase.returnUsersFromGroup(groupName);
        if (users != null) {
            assertTrue(users.length > 0, "Users should be returned for the group.");
        } else {
            System.out.println("No users found for the group or insufficient permissions.");
        }
    }

    @Test
    void testAddAccessForGroup() {
        String groupName = "AccessGroup";
        int userId = App.getUserId(App.getUsername());
        boolean isAccessAdded = ArticleDatabase.addAccessForGroup(userId, groupName);
        assertTrue(isAccessAdded, "Access should be added successfully for the user in the
group.");
    }

    @Test
    void testDeleteAccessForGroup() {
        String groupName = "AccessGroup";
        int userId = App.getUserId(App.getUsername());
        boolean isAccessDeleted = ArticleDatabase.deleteAccessForGroup(userId, groupName);
        assertTrue(isAccessDeleted, "Access should be deleted successfully for the user in the
group.");
    }
```

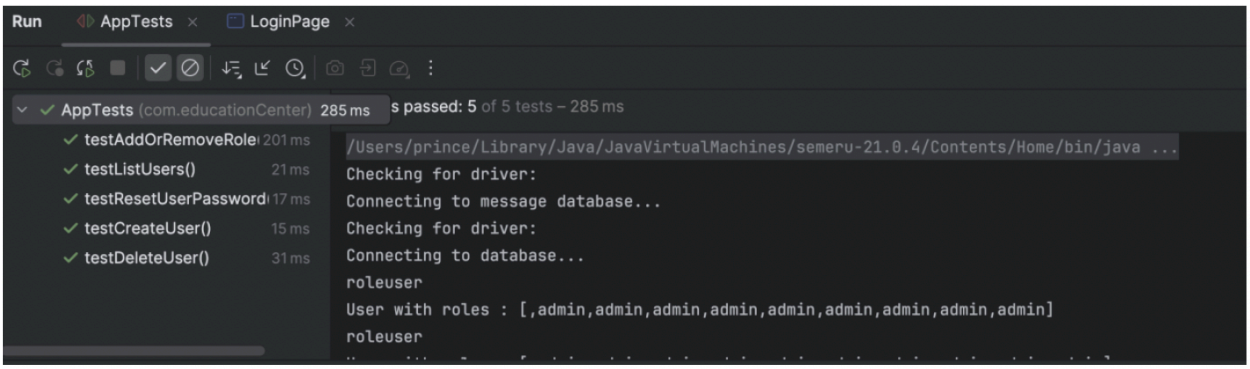
```
    @Test
    void testSearchArticles() {
        String query = "Sample";
        String level = "Intermediate";
        String group = "TestGroup";
        String[][] results = ArticleDatabase.searchArticles(query, level, group);
        assertNotNull(results, "Search results should not be null.");
        assertTrue(results.length > 0, "At least one article should match the search criteria.");
    }

    @Test
    void testIsCurrentAdminOf() {
        String groupName = "AdminGroup";
        boolean isAdmin = ArticleDatabase.isCurrentAdminof(groupName);
        assertFalse(isAdmin, "User should not be an admin for this group by default.");
    }

    @Test
    void testAddAndRemoveInstructorRights() {
        String groupName = "InstructorGroup";
        int instructorId = App.getUserId(App.getUsername());

        // Add rights
        boolean isAdded = ArticleDatabase.addGroupToSpecialAccess(groupName, instructorId);
        assertTrue(isAdded, "Instructor rights should be added successfully.");

        // Remove rights
        boolean isRemoved = ArticleDatabase.removeGroupFromSpecialAccess(groupName,
App.getUsername());
        assertTrue(isRemoved, "Instructor rights should be removed successfully.");
    }

    @Test
    void testBackupAndRestoreArticles() {
        String backupFilename = "BackupFile";
        ArticleDatabase.backupToFile(backupFilename);

        // Verify backup creation in logs or database checks
        ArticleDatabase.callLoadFile(backupFilename);

        // Verify that articles are restored properly
    }
}
```

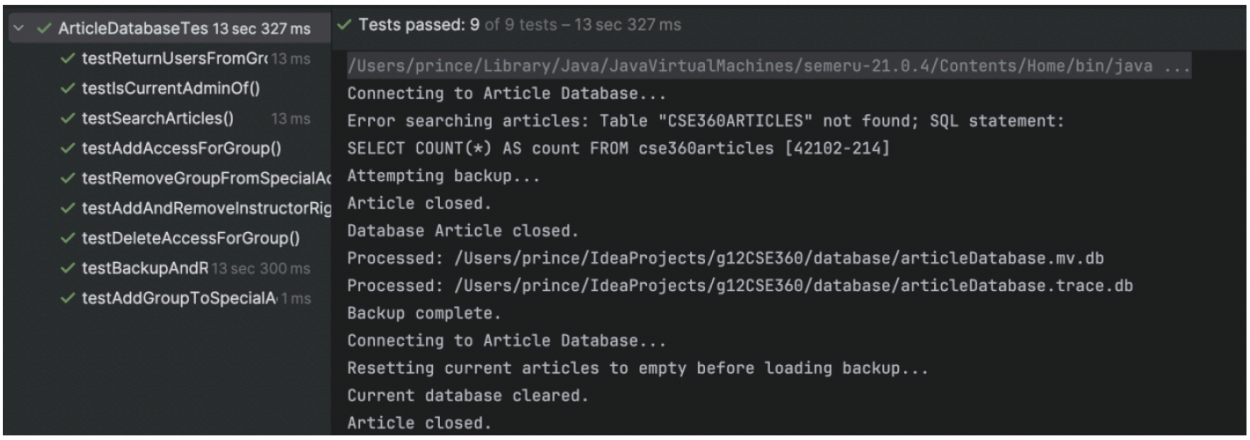
```
package com.educationCenter;

import com.educationCenter.User_Database_Handler.UserDatabaseHelper;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.educationCenter.App.USER_DATABASE_HELPER;
import static org.junit.jupiter.api.Assertions.*;

class AppTests {

    @BeforeEach
    void setUp() throws Exception {
        App.connect();
    }

    @AfterEach
    void tearDown() throws Exception {
        USER_DATABASE_HELPER.closeConnection();
    }

    @Test
    void testCreateUser() throws Exception {
        // Arrange
        String email = "testuser3@example.com";
        String role = "student";

        // Act
        boolean isCreated = App.createUser(email, role);

        // Assert
        assertTrue(isCreated, "The user should be created successfully.");
    }

    @Test
    void testResetUserPassword() throws Exception {
        // Arrange
        String username = "testuser";
        // Simulate user creation
        App.createUser(username + "@example.com", "student");

        // Act
        boolean isReset = App.userResetEmail(username);
```

```
    // Assert
    assertTrue(isReset, "The password reset should succeed.");
}

@Test
void testDeleteUser() throws Exception {
    // Arrange
    String username = "testuser";
    App.createUser(username + "@example.com", "student");

    // Act
    boolean isDeleted = App.deleteUser(username);

    // Assert
    assertTrue(isDeleted, "The user should be deleted successfully.");
}

@Test
void testListUsers() throws Exception {
    // Arrange
    App.createUser("user1@example.com", "admin");
    App.createUser("user2@example.com", "student");

    // Act
    String[][] users = App.listOfUser();

    // Assert
    assertNotNull(users, "The user list should not be null.");
    assertTrue(users.length >= 2, "At least two users should be listed.");
}

@Test
void testAddOrRemoveRole() throws Exception {
    // Arrange
    String username = "roleuser";

    // Act: Add a role
    boolean isRoleAdded = App.addOrRemoveRole(username, "add", "admin");

    // Act: Remove a role
    boolean isRoleRemoved = App.addOrRemoveRole(username, "remove", "student");

    // Assert
```

```
        assertTrue(isRoleAdded, "Admin role should be added successfully.");
        assertTrue(isRoleRemoved, "Student role should be removed successfully.");
    }
}
```

Results Screenshots:





Requirement 10: Admins Can Log Out

Test Type: Manual Test

-Logging out involves clicking a button in the UI, so it's best tested manually.

<u>Setup:</u>

-Log in as an Admin and navigate to the Admin Homepage.

<u>Inputs:</u>

-Click the Logout button.

<u>Outputs:</u>

-The system should redirect to the login page, and the session should be cleared.

-App.getActiveRole() should be set to null.

<u>Assessment:</u>

-This test passes if the login page is displayed and attempts to navigate back to the Admin Homepage fail. It fails if the session persists or the login page isn't displayed.
Coverage:
-This test covers the basic logout functionality. Additional edge cases include testing logout during an active session timeout.

Requirement 11: Users Can Use an Invite Code to Create an Account
Test Type: Manual Test
-Entering an invite code and navigating to the account creation page requires UI interaction, so manual testing works best.
Setup:
-Add a valid invite code (e.g., ABCD1234) to the database.
Inputs:
-Email: testuser@example.com
-Invite Code: ABCD1234
Outputs:
-The system should navigate to the account creation page, allow the user to create an account, and then redirect to the login page.
Assessment:
-This test passes if the account is successfully created and the user can log in with the new credentials. It only works if the invite code is accepted or the account creation fails.
Coverage:
This test covers a valid invite code. Additional tests could handle cases where the code is expired or invalid.

Requirement 12: Users Must Log In to Access Role-Specific Pages
Test Type: JUnit Test
-Role-specific page redirection can be tested programmatically using assertions.
Setup:
-Add users with different roles (Admin, Instructor, Student) to the database.
-Connect the application to the database.
Inputs:
-Username: adminUser, Password: adminPass
-Username: instructorUser, Password: instrPass
-Username: studentUser, Password: studPass
Outputs:
-adminUser should be redirected to the Admin Homepage.
-instructorUser should be redirected to the Instructor Homepage.
-studentUser should be redirected to the Help Page.
Assessment:
The test passes if assertions confirm the correct homepage redirection:
assertEquals("admin", App.appLogin("adminUser", "adminPass"));
assertEquals("instructor", App.appLogin("instructorUser", "instrPass"));
assertEquals("student", App.appLogin("studentUser", "studPass"));

Coverage:
-Tests for each role cover this functionality. Additional tests could check for invalid login attempts.

Requirement 14

14. Each article is given a unique long integer identifier when created, so duplicates can be easily detected.

Setup: This requirement demonstrates the functionality of an internal method and therefore can be accomplished with a JUnit test. The framework of this test was already existing in the system, and a JUnit test was created to utilize these existing functions.

Inputs: This requirement has more to do with the result of creating each article, and is not dependent on a particular set of input values. Therefore for the inputs we can just use placeholder data. This particular test is testing the functionality of the createArticle() function which accepts a list of

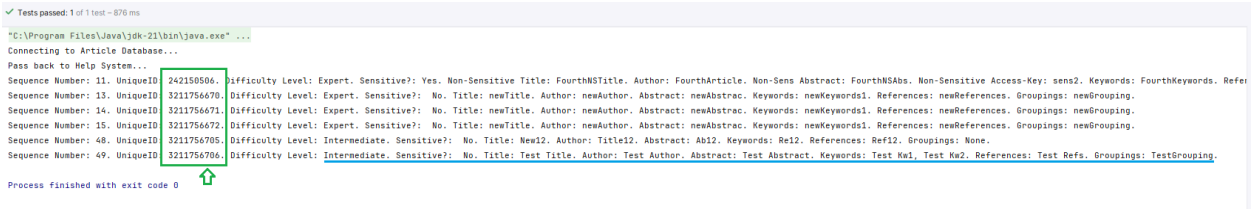
Outputs: The expected output for testing requirement is simply that there is a uniqueID that has been created that is associated with each help article item. This can be verified by displaying the articles that have been created, looking at the uniqueID field, and ensuring that this field has been created and is in fact unique for each article.

Assessments: As touched on above, testing this requirement involves testing the createArticle() function, as well as calling the displayArticle() function so we can verify that the uniqueID has been created correctly. Therefore, this JUnit test calls both of these functions using placeholder data that can be used to verify it is functioning correctly.

Coverage: By calling these functions explained above, and with existing article database data in the system, together we can see that the unique identifier is in fact created, and can be compared to other existing article uniqueIDs. Together, this explains how this single JUnit test allows us to know this requirement is indeed working as intended.

Test Suite Submission: Below are screenshots showing the code of the JUnit test itself, as well as the output resulting from that test. The screenshot of the code shows the approach explained above, and the output screenshot shows how the newly created article item does in fact have a new uniqueID (annotated in green), and this article has been created with the placeholder data in the JUnit test (annotated in blue).

```java
class Req14 {
    @Test
    void createArticle() throws Exception {
        // Instantiate Article DB For Test
        ArticleDatabase helpDB = new ArticleDatabase(); String[] simArgs ={"arg1", "arg2", "arg3"};
        try { helpDB.main(simArgs); } catch (Exception e) { throw new RuntimeException(e); }
        // Create Placeholder Test Data
        String newTitle = "Test Title"; String newBody = "Test Body";
        String newAuthor = "Test Author"; String newAbstrac = "Test Abstract";
        String newKeywords = "Test Kw1, Test Kw2"; String newReferences = "Test Refs"; String newDifficulty = "2";
        String newGrouping = "TestGrouping";
        // Call Relevant Functions To Verify Functionality
        helpDB.callCreateArticle(newTitle, newBody, newAuthor, newAbstrac, newKeywords, newReferences, newDifficulty, newGrouping);
        helpDB.callDisplayArticles();
    }
}
```



Requirement 15

15. The admin and the instruction team roles are enhanced by providing commands to back up and restore the help system data to an admin/instructor-named external data file.

Setup: This requirement demonstrates the functionality of the backup feature existing in the system, which can be accomplished by creating a JUnit test with the placeholder user-specified filename, and code to call the backup function.

Inputs: As this requirement involves the use of the backup feature, in particular allowing for a specified user-named external data file, this input for this test includes this user-specified backup filename.

Outputs: The expected output for this requirement is that a backup file is created with the expected data contained within. This can be referenced by console/debug output, as well as by viewing the filesystem to verify that this backup has been correctly created.

Assessments: As touched on above, testing this requirement involves using a user-specified filename to then call the backup function. From here, we can view console debug output that verifies that a backup has been created. Additionally, viewing the running computer's filesystem can be done to verify that the backup has been generated with the correct specified filename.

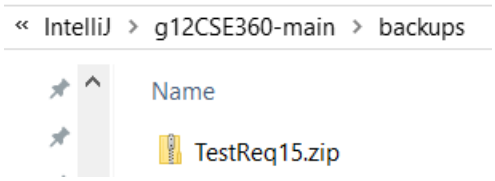
Coverage: By calling the backup function with a user-specified filename, we are able to see with the console debug output as well as by viewing the filesystem that this backup file was created correctly. Because this is a fairly straight forward action, with the only variable being a different string used for the filename, this JUnit test can be used to verify this requirement is met.

Test Suite Submission: Below are screenshots showing the code of the JUnit test itself, as well as the output resulting from that test. The screenshot of the code shows the approach explained above,

and the output screenshot the console debug output showing success. Additionally, I have included a screenshot showing the filesystem to further verify that a backup was created, and was created with the specified filename.

```java
class Req15 {
    @Test
    void backupToFile() throws SQLException {
        // Instantiate Article DB For Test
        ArticleDatabase helpDB = new ArticleDatabase(); String[] simArgs ={"arg1", "arg2", "arg3"};
        try { helpDB.main(simArgs); } catch (Exception e) { throw new RuntimeException(e); }
        // Create Placeholder Test Data
        String backupFilename = "TestReq15";
        // Call Relevant Functions To Verify Functionality
        helpDB.databaseHelper.backupToFile(backupFilename);
    }
}
```

```
✓ Tests passed: 1 of 1 test – 865 ms

"C:\Program Files\Java\jdk-21\bin\java.exe" ...
Connecting to Article Database...
Pass back to Help System...
Attempting backup...
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.mv.db
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.trace.db
Connecting to Article Database...
Backup completed.

Process finished with exit code 0
```



Requirement 16

> 16. An option is provided to remove all the existing help articles or merge the backed-up copies with the current help articles. When the unique long integer identifiers match, the backed-up copy will not be added.

Setup: This requirement test demonstrates the functionality of the load from backup feature where article database items with a matching unique identifier field are skipped and not merged with the current database. Because this has involves not just the internal code of the help system, but also conditions and contents of external backup files, this requirement is demonstrated using a manual test.

Inputs: As this requirement involves the use of the backup feature, the input for this test includes a user-specified backup filename. Which in this case will be a backup file that contains some database articles that include a matching unique identifier to those currently in the running database.

Outputs: The expected output for this requirement after a backup file is attempted to be loaded is that after this backup load is completed, upon viewing the newly updated current database, we will only see article database items added to the current database which did not have a matching unique identifier to any existing database items. This can be referenced by viewing console/debug output, as well as by doing fresh loads from these backup files to verify they contained data that should have been excluded from this manual test.

Assessments:

Current Database:

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. I
Sequence Number: 13. UniqueID: 3211756670.
Sequence Number: 14. UniqueID: 3211756671.
```

Contents Of Backup File To Load:

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. I
Sequence Number: 13. UniqueID: 3211756670.
Sequence Number: 14. UniqueID: 3211756671.
Sequence Number: 15. UniqueID: 3211756672.
Sequence Number: 48. UniqueID: 3211756705.
Sequence Number: 49. UniqueID: 3211756706.
Sequence Number: 50. UniqueID: 3211756707.
```

In the screenshots above we can see the contents of the current running database as well as the contents of a saved database backup file. We can see that some of the items in the saved backup file have a matching unique identifier as the articles in the existing database. So, in this test, for a valid test result, we would expect the only items which do not have a matching unique ID to be merged. Below is a screenshot of the console/debug output from running the test under this condition, where we can see that it performs as expected, merging articles, but skipping those with matching identifiers:

```
Merge database from file.
Enter the .zip database path and filename (with extension): C:\IntelliJ\g12CSE360-main\backups\Req3_1.zip

Article 242150506 already exists, skip merge.
Article 3211756670 already exists, skip merge.
Article 3211756671 already exists, skip merge.
Article 3211756672 merged.
Article 3211756705 merged.
Article 3211756706 merged.
Article 3211756707 merged.
```

Coverage: Because this requirement deals with the merging of current database articles and those from a backup file, and it specifically only handles logic pertaining to the unique ID field, this test demonstrates that this requirement is satisfied.

3. Requirement 17

17. The system shall support a mechanism to support multiple groups of help articles. Grouped articles can be backed up, so only the articles in the group are in the backup.

Setup: This requirement demonstrates the functionality of the backup feature existing in the system, though it differs from the previous requirement test in that this requirement specifies to only back up articles that exist within a specific group. Because this test will involve verifying the data in this backup was created correctly, it will be done using a manual test.

Inputs: As this requirement involves the use of the backup feature, in particular allowing for a specified user-named external data file as well as a user-specified grouping identifier for which to only backup articles of that group, the two input values necessary will be the specified backup data file, and the string value of the group that is to be backed up.

Outputs: The expected output for this requirement is that a backup file is created with the expected data contained within. This can be referenced by console/debug output, viewing the filesystem to verify that this backup has been correctly created, as well as utilizing the load/restore feature to load that newly generated backup into the current database, where it can be displayed to verify the contents of the backup file contain only those within that group.

Assessments: As touched on above, because this requirement involves verifying that the data contained within the generated backup is only articles belonging to a certain group, a load operation will be performed, which will then be displayed, so it can be verified that this new database backup file contains only the correct items.

Coverage: Because this test involves generating a backup, and ensuring that the backup created fits the specifications, we can see from the result of the manual test that indeed this has been carried out. With the only variance possible being the string filename to save, and the string grouping identifier, this manual test will demonstrate this requirement is fulfilled.

Test Suite Submission: Below are screenshots console debug output for each of the steps for this test. This includes showing the original database contents (which include articles with various grouping identifiers), running the backup function with the values explained above, restoring from this new backup file, and the new full database display function where we can see that this new backup file did in fact only contain articles from the specified grouping.

Original Database:

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle. Non-Sens Abstract: FourthNSAbs. Non-Sensitive Access-Key: sens2. Keywords: FourthKeywords.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 15. UniqueID: 3211756672. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 48. UniqueID: 3211756705. Difficulty Level: Intermediate. Sensitive?:  No. Title: New12. Author: Title12. Abstract: Ab12. Keywords: Re12. References: Ref12. Groupings: None.
Sequence Number: 49. UniqueID: 3211756706. Difficulty Level: Intermediate. Sensitive?:  No. Title: Test Title. Author: Test Author. Abstract: Test Abstract. Keywords: Test Kw1, Test Kw2. References: Test Refs. Groupings: TestGrouping.
```

```
Enter the backup filename (without file extension): TestReq17

Enter Groupings String To Backup By: TestGrouping
Attempting backup...
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.mv.db
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.trace.db
Connecting to Article Database...
Backup completed.
Article deleted.
Article deleted.
Article deleted.
Article deleted.
Article deleted.
Attempting backup...
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.mv.db
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.trace.db
Connecting to Article Database...
Backup completed.
Resetting current articles to empty before loading backup:
Article 49 set to empty.
There are no articles in the database.
Existing database disconnected, backup database loaded from file.
Connecting to Article Database...
```

```
Load database from file.
Enter the .zip database path and filename (with extension): C:\IntelliJ\g12CSE360-main\backups\TestReq17.zip

Resetting current articles to empty before loading backup:
Article 11 set to empty.
Article 13 set to empty.
Article 14 set to empty.
Article 15 set to empty.
Article 48 set to empty.
Article 49 set to empty.
There are no articles in the database.
Existing database disconnected, backup database loaded from file.
Connecting to Article Database...
```

New Database Backup Loaded, Containing Articles With Specified Grouping Only:

```
Display All Articles:
Sequence Number: 49. UniqueID: 3211756706. Difficulty Level: Intermediate. Sensitive?:  No. Title: Test Title. Author: Test Author. Abstract: Test Abstract. Keywords: Test Kw1, Test Kw2. References: Test Refs. Groupings: TestGrouping.
```

Requirement 18

18. Both admins and instructors may create, update, view, and delete help articles.

Setup: This requirement test demonstrates the functionality of a number of features available within the help system. Because this requirement deals with a number of different features that are being used together, as well as how this requirement's validity is dependent on the correct expected output of these features, a manual test is used to demonstrate this requirement is met.

Inputs: As this requirement deals with the creation, deletion, updating, and viewing of help articles, specifically, the required input for this test includes a complete set of test article data which can be used to add a new article to the system.

Outputs: There is a number of different expected outputs for this requirement. After creating a new article, we expect that this new article is to be contained within the current help system, and therefore would be visible after displaying the database. Also, for the updating of articles, we expect that the article update has been correctly performed, and therefore in a subsequent request to display the article database contents, we expect that this output shows these changes have been made. Additionally, for the delete feature, we would expect that after the delete operation has been performed, that this article that has been selected to be deleted is no longer existing within the current database.

Assessments:

For the manual test demonstrating this test requirement is satisfied, we begin with an existing article database which we can view with console/debug output showing us its original contents that expect to see modified after the actions specified in this requirement have been carried out. This original database is shown in the screenshot below (truncated as to fit on this page, but showing enough information that we can deduce a correct or incorrect handling of these operations):

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac.
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac.
```

Now in the below screenshot we can see the console/debug output of the article database contents being displayed after a new article has been added. We can see it has been added at the bottom of the list, with the newly generated sequence number and unique identifier.

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Auth
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abs
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abs
Sequence Number: 51. UniqueID: 3211756708. Difficulty Level: Expert. Sensitive?:  No. Title: Req5_Test. Author: Req5_Author.
```

After the step taken shown in the previous screenshot, we can now see the console/debug output of the article database contents being displayed after they have had a database request to update/modify that new article's contents. The screenshot is truncated but enough for us to see changes have been made which can be seen in the bottommost article's Title and Author fields.

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac.
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac.
Sequence Number: 51. UniqueID: 3211756708. Difficulty Level: Advanced. Sensitive?:  No. Title: Req5_Updated_Title. Author: Req5_Updated_Author.
```

Finally, to complete testing of the functionality described in this requirement's testing, below we have a screenshot showing the console/debug output of the article database contents being displayed after a request has been made to the database for a certain article to be deleted. We can see in the newly updated article database contents that indeed this article which existed before no longer exists.

```
Delete article.
Enter sequence number to delete: 14
Article deleted.

Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac.
Sequence Number: 51. UniqueID: 3211756708. Difficulty Level: Advanced. Sensitive?:  No. Title: Req5_Updated_Title. Author: Req5_Updated_Author.
```

Coverage: This specific requirement was a combination of a number of different features pertaining to the user interaction with the article database. Specifically, this included viewing, adding a new article, updating an existing article, and deletion of an existing article. Each of these features can be seen to be functioning as intended with the correct expected output in this tests, referenced by the screenshots provided and the accompanying explanation.

Requirement 19

19. Both admins and instructors may list all the help articles and subsets of the help articles in a group or multiple groups.

Setup: This requirement demonstrates the functionality of a specific display feature, specifically this requirement is unique in that it allows users to enter in a user-defined string that represents articles belonging only specific multiple groups to be displayed. Therefore we can use a manual test to fully demonstrate the article database meets the functionality for this requirement.

Inputs: As this requirement involves the use of a user-specified string which represents multiple grouping identifiers, which in this case are space-separated. Therefore, as this test differs from other requirements in the fact it is searching for only articles matching multiple grouping identifiers, this test will include a space-separated test string containing the space-separated identifiers of two unique article groups.

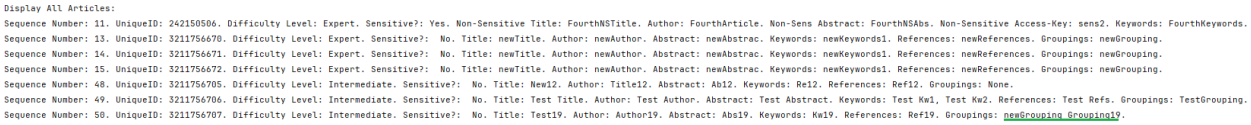
Outputs: The expected output for this requirement is that a subset of the article database is displayed, specifically with only articles that are matching both of the grouping identifiers included in the user-specified input string. This differs from other requirements where a more broad list of articles are returned from the database.

Assessments: For this requirement, we can see by comparing the standard display all articles feature, to this feature of returning articles only matching a specified set of multiple groupings. We can verify that this works by seeing that articles not matching the criteria are not returned, and only articles matching these multiple groupings are displayed.

Coverage: Because this requirement is handled by database architecture allowing for space-separated strings, we can see that this test verifies that the article database system has the ability to display only articles matching multiple grouping identifiers.

Test Suite Submission: Below are screenshots showing the complete article database contents first (with the article matching the multiple grouping identifier criteria annotated in green), then followed by the console debug output showing the result of the display function called with a space-separated string consisting of multiple grouping identifiers. We can see that only a subset of the database is displayed, and that the subset displayed correctly matches the criteria, as it is a member of both grouping identifiers.

All Articles In Current Database:



The Full Output Of the Resulting Multiple-Grouping Identifier Display Request:

```
Display articles by grouping.
Enter group IDs (space-separated): newGrouping Grouping19

Unique Header: Difficulty: 2 | Groupings IDs: newGrouping Grouping19 | Sensitive?: No
Sequence Number: 50. Unique ID: 3211756707.
Title: Test19.
Author(s): Author19.
Keywords: Kw19.
Abstract: Abs19.
Body: Body19.
References: Ref19.
End Article 50 Data.
```

Requirement 30

> 30. The student and instructor role is enhanced with additional commands: Set the content level of the articles to be returned (e.g., beginner, intermediate, advanced, expert). The student may also specify an "all" level to see all the articles independent of the level. The default is "all".

Setup: This requirement test demonstrates the functionality of the display feature made available to the student and instructor user roles within their respective user interfaces. In this case, these different user types are given the same article database function to handle this as it is the same expected functionality. To demonstrate that this test's requirements are satisfied, this requirement will be demonstrating using manual testing.

Inputs: As this requirement has to do with the user selection of the article database display feature to return only articles matching a user specified content level, this content level will be used as input for each of the various test cases that are demonstrated in this requirement's testing suite.

Outputs: The expected output for each of the specified content levels is that we are given an output displaying each of the articles in the current database that meets this current criteria. We can verify the behavior of each of these conditions of the display articles feature by examining the console/debug output.

Assessments:

For this requirement we first show the article database output when there is no content level specified, meaning the default behavior is set to “All” where all articles are displayed. We can see from this screenshot below showing this behavior also what we expect from the subsequent tests for this test requirement. We can see we have a number of articles, including articles from a number of content levels, all displayed together. This indeed verifies the default content level of “All” as working correctly.

```
Unique Header: Difficulty: Expert | Groupings IDs: None | Sensitive?: Yes, Non-Sensitive Title: |
Sequence Number: 11. Unique ID: 242150506.
Title: FourthArticleSens.
Author(s): FourthArticle.
Keywords: FourthKeywords.
Abstract: FourthAbstractSens.
Body: FourtArticleBody.
References: FourthReferences.
End Article 11 Data.

Unique Header: Difficulty: Expert | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.

Unique Header: Difficulty: Advanced | Groupings IDs: Req5_Updated_Grouping | Sensitive?: No
Sequence Number: 51. Unique ID: 3211756708.
Title: Req5_Updated_Title.
Author(s): Req5_Updated_Author.
Keywords: Req5_Updated_Keywords.
Abstract: Req5_Updated_Abs.
Body: Req5_Updated_Body.
References: Req5_Updated_Refs.
End Article 51 Data.

Unique Header: Difficulty: Beginner | Groupings IDs: TestGrps | Sensitive?: No
Sequence Number: 52. Unique ID: 3211756709.
Title: TestArticle.
Author(s): TestAuthor.
Keywords: TestKeyword.
Abstract: TestAbstract.
Body: TestBody.
References: TestRefs.
End Article 52 Data.
```

Next, in the screenshot below we show the behavior of this same article display feature, but when given the input from the user to display articles only matching the content level of “Beginner”. We can see from the screenshot showing the complete console/debug output from this operation, including only articles which match this criteria. This is working as expected.

```
Display Articles By Difficulty: Beginner

Unique Header: Difficulty: Beginner | Groupings IDs: TestGrps | Sensitive?: No
Sequence Number: 52. Unique ID: 3211756709.
Title: TestArticle.
Author(s): TestAuthor.
Keywords: TestKeyword.
Abstract: TestAbstract.
Body: TestBody.
References: TestRefs.
End Article 52 Data.
```

Now, in the screenshot below we show the behavior of this same article display feature, but when given the input from the user to display articles only matching the content level of “Intermediate”. We can see from the screenshot showing the complete console/debug output from this operation, including only articles which match this criteria, which in this case as we can see from the first screenshot in this test suite, is none. This is working as expected.

```
Display Articles By Difficulty: Intermediate
```

Now, in the screenshot below we show the behavior of this same article display feature, but when given the input from the user to display articles only matching the content level of “Advanced”. We can see from the screenshot showing the complete console/debug output from this operation, including only articles which match this criteria. This is working as expected.

```
Display Articles By Difficulty: Advanced

Unique Header: Difficulty: Advanced | Groupings IDs: Req5_Updated_Grouping | Sensitive?: No
Sequence Number: 51. Unique ID: 3211756708.
Title: Req5_Updated_Title.
Author(s): Req5_Updated_Author.
Keywords: Req5_Updated_Keywords.
Abstract: Req5_Updated_Abs.
Body: Req5_Updated_Body.
References: Req5_Updated_Refs.
```

Now, in the screenshot below we show the behavior of this same article display feature, but when given the input from the user to display articles only matching the content level of “Expert”. We can see from the screenshot showing the complete console/debug output from this operation, including only articles which match this criteria. This is working as expected.

```
Display Articles By Difficulty: Expert

Unique Header: Difficulty: Expert | Groupings IDs: None | Sensitive?: Yes, Non-S
Sequence Number: 11. Unique ID: 242150506.
Title: FourthArticleSens.
Author(s): FourthArticle.
Keywords: FourthKeywords.
Abstract: FourthAbstractSens.
Body: FourtArticleBody.
References: FourthReferences.
End Article 11 Data.

Unique Header: Difficulty: Expert | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.
```

Coverage: Because this requirement deals with the displaying of articles that meet only the criteria of matching a certain content level (with the default content level of “All) we can see from the above test cases/screenshots that all of these content level requests are working as intended, including with all possible database states including this data which are no matching articles one matching article, or many matching articles. Therefore we can see from these set of tests that this requirement is satisfied.

Requirement 31

> 31. The student and instructor role is enhanced with additional commands: Specify a group of articles to limit returned articles to just those items in a group (e.g., a group may be help articles for a particular assignment).

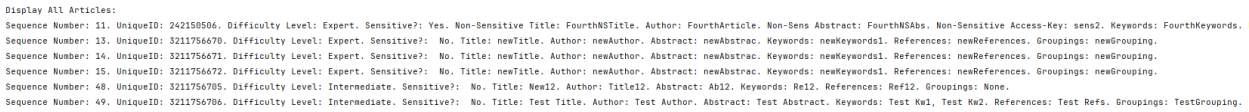
Setup: This requirement demonstrates the ability for the article database to return only article items with a user-specified grouping identifier. Because of this, we can utilize existing functions contained within the article database class that will allow this to be seen.

Inputs: The input for this requirement will be the user-specified grouping identifier which will be used to call the appropriate database function. This input string will allow the database to then return only the articles which are a part of that specified group.

Outputs: The expected output for this requirement is a parsed down list of the articles contained within the database, specifically, with only articles matching the given input string for the grouping identifier. This output will demonstrate the functionality works as intended.

Assessments:

The complete database contents:



Now displaying only those matching the test string “newGroupings” to represent the requested grouping identifier. This is the complete output of this feature, and the output is correct as expected, displaying only a subset of the overall database contents.

```
Display articles by grouping.
Enter group IDs (space-separated): newGrouping

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 14. Unique ID: 3211756671.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 14 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 15. Unique ID: 3211756672.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 15 Data.
```

Coverage: Because the differing potential inputs are just different strings that may match existing articles grouping field, this test will demonstrate the functionality of this requirement.

Requirement 33

The student and instructor role is enhanced with additional commands: When a search is performed, the system will display the following.
- The first line displayed specifies the group that is currently active,
- The number of articles that match each level,
- A list of articles matching the search criteria in a short form. The short form includes a sequence number (1 - n), the title, the author(s), and the abstract.

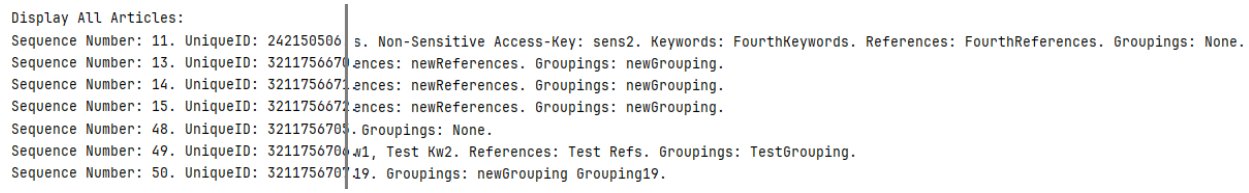
Setup: This requirement demonstrates the ability for the article database to return a collection of article database items with in a specific format after being called for displaying a certain article group. This requirement clearly defines the format for this article group display output, and can be demonstrated as working with a manual test.

Inputs: The input for this requirement will be the user-specified grouping identifier which will be used to call the appropriate database function. This input string will allow the database to then return only the articles which are a part of that specified group, and with the specific format required.

Outputs: The expected output for this requirement is a parsed down list of the articles contained within the database, specifically, with only articles matching the given input string for the grouping identifier. Additionally, the data contained within this article group display includes a first-line header with the group identifier and the number of matching articles, before showing a lit of each article in a short form that includes the sequence number, the title, author, and the abstract.

Assessments:

The complete database contents are shown in the screenshot below (this screenshot has been cropped as to fit into this document and displaying the relevant data). From this we can see the current article database running during the test included a number of articles, with only a certain subgroup of the articles that match the grouping identifier which will be requested and shown in the subsequent screenshots:



Next for this test, included in the below screenshot is the output of the article database feature that relates to this requirement. We can see that this output fits the format of the requirement and also contains the correct articles, with the correct respective data for each. We can see when referencing the above screenshot of the entire article database, that the below list is indeed the full list of articles which should be returned when using the specified grouping identifier. We can also see in the below screenshot that the article count is correctly displayed, along with each article's short-form display including its sequence number, title, author and abstract.

```
Group: newGrouping | Article Count: 4.

Sequence Number: 13
Title: newTitle.
Author(s): newAuthor.
Abstract: newAbstrac

Sequence Number: 14
Title: newTitle.
Author(s): newAuthor.
Abstract: newAbstrac

Sequence Number: 15
Title: newTitle.
Author(s): newAuthor.
Abstract: newAbstrac

Sequence Number: 50
Title: Test19.
Author(s): Author19.
Abstract: Abs19
```

Coverage: This requirement, which handles returning a subgroup of articles based on a user specified identifier can be seen from the included test case that it works. From this test we can see that the article count is tallied correctly, and the correct data is retrieved from the database. From this, we can conclude that this requirement is met by the help system.

Requirement 34

34. The instructor role is enhanced with additional commands: Instructors may specify a group or a special access group to reduce the length of returned items.

Setup: This requirement demonstrates the ability for the article database to return only article items with a user-specified grouping identifier. Because of this, we can utilize existing functions contained within the article database class that will allow this to be seen.

Inputs: The input for this requirement will be the user-specified grouping identifier which will be used to call the appropriate database function. This input string will allow the database to then return only the articles which are a part of that specified group.

Outputs: The expected output for this requirement is a parsed down list of the articles contained within the database, specifically, with only articles matching the given input string for the grouping identifier. This output will demonstrate the functionality works as intended.

Assessments: As touched on above, because the only differing values possible for this requirement would be different input strings for the requested grouping identifier, this manual test can show that by calling the appropriate function with an input string representing this grouping, that the requirement is carried out successfully.

Coverage: Once again, because the differing potential inputs are just different strings that may match existing articles grouping field, this test will demonstrate the functionality of this requirement.

Test Suite Submission: Below are screenshots showing console debug output for each of the steps for this test. This includes showing the original database contents (which include articles with various grouping identifiers), as well as the result of the display function after calling it with a specified string to return only those matching that grouping identifier. We can see this display only a particular grouping feature works as intended and fulfills this requirement.

The complete database contents:

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle. Non-Sens Abstract: FourthNSAbs. Non-Sensitive Access-Key: sens2. Keywords: FourthKeywords.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 15. UniqueID: 3211756672. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 48. UniqueID: 3211756705. Difficulty Level: Intermediate. Sensitive?:  No. Title: New12. Author: Title12. Abstract: Ab12. Keywords: Re12. References: Ref12. Groupings: None.
Sequence Number: 49. UniqueID: 3211756706. Difficulty Level: Intermediate. Sensitive?:  No. Title: Test Title. Author: Test Author. Abstract: Test Abstract. Keywords: Test Kw1, Test Kw2. References: Test Refs. Groupings: TestGrouping.
```

Now displaying only those matching the test string “newGroupings” to represent the requested grouping identifier. This is the complete output of this feature, and the output is correct as expected, displaying only a subset of the overall database contents.

```
Display articles by grouping.
Enter group IDs (space-separated): newGrouping

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 14. Unique ID: 3211756671.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 14 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 15. Unique ID: 3211756672.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 15 Data.
```

Requirement 36

36. The instructor role is enhanced with additional commands: Instructors may create, view, edit, and delete help articles.

Setup: This requirement demonstrates the ability for the article database to return only article items with a user-specified grouping identifier. Because of this, we can utilize existing functions contained within the article database class that will allow this to be seen.

Inputs: The input for this requirement will be the user-specified grouping identifier which will be used to call the appropriate database function. This input string will allow the database to then return only the articles which are a part of that specified group.

Outputs: The expected output for this requirement is a parsed down list of the articles contained within the database, specifically, with only articles matching the given input string for the grouping identifier. This output will demonstrate the functionality works as intended.

Assessments: As touched on above, because the only differing values possible for this requirement would be different input strings for the requested grouping identifier, this manual test can show that by calling the appropriate function with an input string representing this grouping, that the requirement is carried out successfully.

Coverage: Once again, because the differing potential inputs are just different strings that may match existing articles grouping field, this test will demonstrate the functionality of this requirement.

Test Suite Submission: Below are screenshots showing console debug output for each of the steps for this test. This includes showing the original database contents (which include articles with various grouping identifiers), as well as the result of the display function after calling it with a specified string to return only those matching that grouping identifier. We can see this display only a particular grouping feature works as intended and fulfills this requirement.

The complete database contents:

```
Display All Articles:
Sequence Number: 11. UniqueID: 242150506. Difficulty Level: Expert. Sensitive?: Yes. Non-Sensitive Title: FourthNSTitle. Author: FourthArticle. Non-Sens Abstract: FourthNSAbs. Non-Sensitive Access-Key: sens2. Keywords: FourthKeywords.
Sequence Number: 13. UniqueID: 3211756670. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 14. UniqueID: 3211756671. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 15. UniqueID: 3211756672. Difficulty Level: Expert. Sensitive?:  No. Title: newTitle. Author: newAuthor. Abstract: newAbstrac. Keywords: newKeywords1. References: newReferences. Groupings: newGrouping.
Sequence Number: 48. UniqueID: 3211756705. Difficulty Level: Intermediate. Sensitive?:  No. Title: New12. Author: Title12. Abstract: Ab12. Keywords: Re12. References: Ref12. Groupings: None.
Sequence Number: 49. UniqueID: 3211756706. Difficulty Level: Intermediate. Sensitive?:  No. Title: Test Title. Author: Test Author. Abstract: Test Abstract. Keywords: Test Kw1, Test Kw2. References: Test Refs. Groupings: TestGrouping.
```

Now displaying only those matching the test string “newGroupings” to represent the requested grouping identifier. This is the complete output of this feature, and the output is correct as expected, displaying only a subset of the overall database contents.

```
Display articles by grouping.
Enter group IDs (space-separated): newGrouping

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 14. Unique ID: 3211756671.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 14 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 15. Unique ID: 3211756672.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 15 Data.
```

Requirement 39

39. Instructors may backup and restore articles and groups of articles.

Setup: This requirement demonstrates the functionality of the backup feature existing in the system, though it differs from the previous requirement test in that this requirement specifies to only back up articles that exist within a specific group. Because this test will involve verifying the data in this backup was created correctly, it will be done using a manual test.

Inputs: As this requirement involves the use of the backup feature, in particular allowing for a specified user-named external data file as well as a user-specified grouping identifier for which to only backup articles of that group, the two input values necessary will be the specified backup data file, and the string value of the group that is to be backed up.

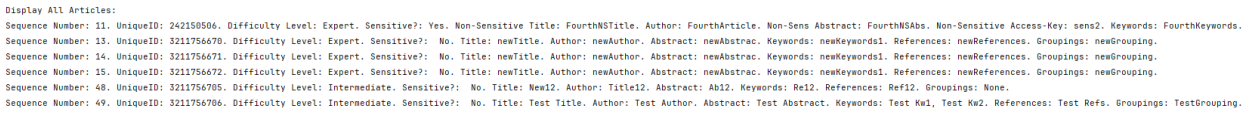
Outputs: The expected output for this requirement is that a backup file is created with the expected data contained within. This can be referenced by console/debug output, viewing the filesystem to verify that this backup has been correctly created, as well as utilizing the load/restore feature to load that newly generated backup into the current database, where it can be displayed to verify the contents of the backup file contain only those within that group.

Assessments: As touched on above, because this requirement involves verifying that the data contained within the generated backup is only articles belonging to a certain group, a load operation will be performed, which will then be displayed, so it can be verified that this new database backup file contains only the correct items.

Coverage: Because this test involves generating a backup, and ensuring that the backup created fits the specifications, we can see from the result of the manual test that indeed this has been carried out. With the only variance possible being the string filename to save, and the string grouping identifier, this manual test will demonstrate this requirement is fulfilled.

Test Suite Submission: Below are screenshots console debug output for each of the steps for this test. This includes showing the original database contents (which include articles with various grouping identifiers), running the backup function with the values explained above, restoring from this new backup file, and the new full database display function where we can see that this new backup file did in fact only contain articles from the specified grouping.

Original Database:




```
Enter Groupings String To Backup By: TestGrouping
Attempting backup...
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.mv.db
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.trace.db
Connecting to Article Database...
Backup completed.
Article deleted.
Article deleted.
Article deleted.
Article deleted.
Article deleted.
Attempting backup...
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.mv.db
Processed: C:/IntelliJ/g12CSE360-main/database/Test2001.trace.db
Connecting to Article Database...
Backup completed.
Resetting current articles to empty before loading backup:
Article 49 set to empty.
There are no articles in the database.
Existing database disconnected, backup database loaded from file.
Connecting to Article Database...
```

```
Resetting current articles to empty before loading backup:
Article 11 set to empty.
Article 13 set to empty.
Article 14 set to empty.
Article 15 set to empty.
Article 48 set to empty.
Article 49 set to empty.
There are no articles in the database.
Existing database disconnected, backup database loaded from file.
Connecting to Article Database...
```

New Database Backup Loaded, Containing Articles With Specified Grouping Only:

```
Display All Articles:
Sequence Number: 49. UniqueID: 3211756706. Difficulty Level: Intermediate. Sensitive?:  No. Title: Test Title. Author: Test Author. Abstract: Test Abstract. Keywords: Test Kw1, Test Kw2. References: Test Refs. Groupings: TestGrouping.
```

Requirement 45

45. Admins may backup and restore articles, groups of articles, and special access groups of articles.

Setup: This requirement demonstrates the ability for the article database to return only article items with a user-specified grouping identifier. Because of this, we can utilize existing functions contained within the article database class that will allow this to be seen.

Inputs: The input for this requirement will be the user-specified grouping identifier which will be used to call the appropriate database function. This input string will allow the database to then return only the articles which are a part of that specified group.

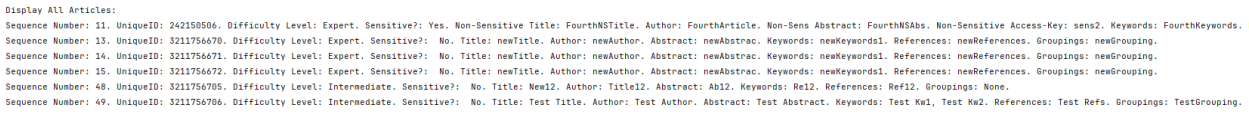
Outputs: The expected output for this requirement is a parsed down list of the articles contained within the database, specifically, with only articles matching the given input string for the grouping identifier. This output will demonstrate the functionality works as intended.

Assessments: As touched on above, because the only differing values possible for this requirement would be different input strings for the requested grouping identifier, this manual test can show that by calling the appropriate function with an input string representing this grouping, that the requirement is carried out successfully.

Coverage: Once again, because the differing potential inputs are just different strings that may match existing articles grouping field, this test will demonstrate the functionality of this requirement.

Test Suite Submission: Below are screenshots showing console debug output for each of the steps for this test. This includes showing the original database contents (which include articles with various grouping identifiers), as well as the result of the display function after calling it with a specified string to return only those matching that grouping identifier. We can see this display only a particular grouping feature works as intended and fulfills this requirement.

The complete database contents:



Now displaying only those matching the test string “newGroupings” to represent the requested grouping identifier. This is the complete output of this feature, and the output is correct as expected, displaying only a subset of the overall database contents.

```
Display articles by grouping.
Enter group IDs (space-separated): newGrouping

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 13. Unique ID: 3211756670.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 13 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 14. Unique ID: 3211756671.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 14 Data.

Unique Header: Difficulty: 4 | Groupings IDs: newGrouping | Sensitive?: No
Sequence Number: 15. Unique ID: 3211756672.
Title: newTitle.
Author(s): newAuthor.
Keywords: newKeywords1.
Abstract: newAbstrac.
Body: newBody.
References: newReferences.
End Article 15 Data.
```

Requirement 2: The student role is enhanced with additional commands: Students may always perform a set of actions, including quitting the application, sending a generic message to the help system, and sending a specific message to the help system. Generic messages are used to express confusion about how to use the tool.

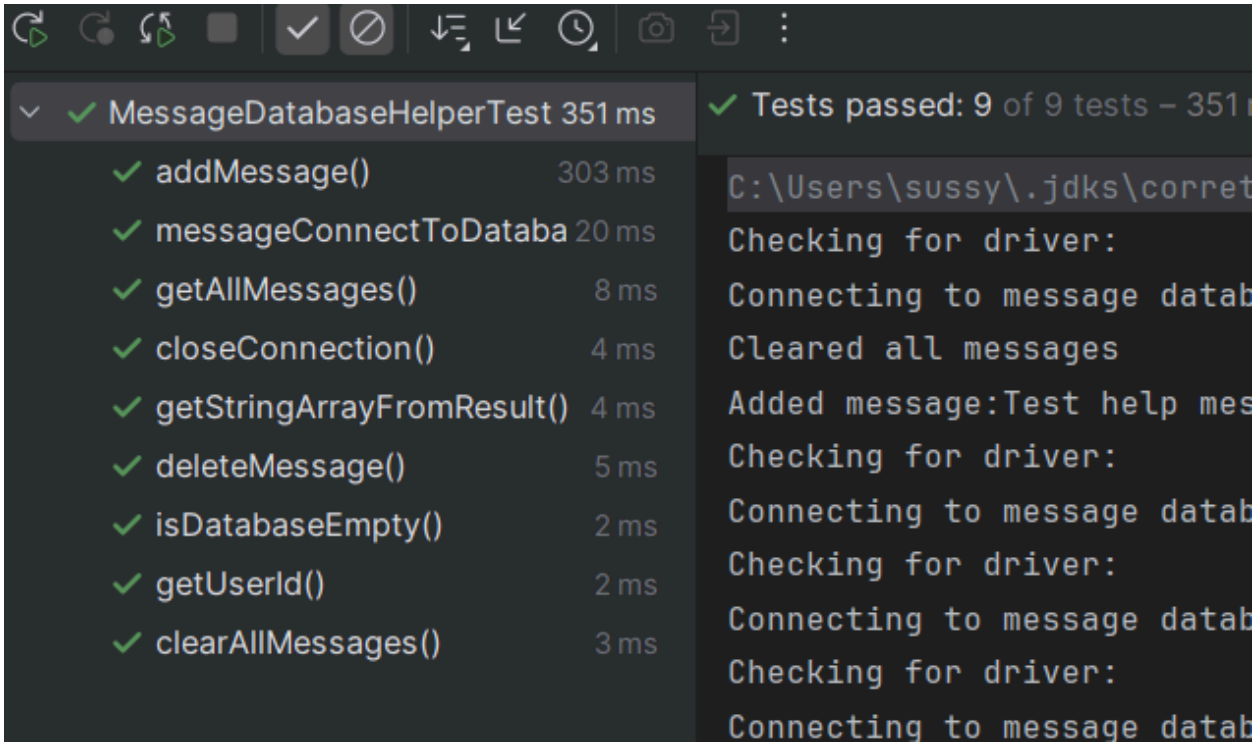
Setup: The setup for testing this database includes the creation of the database helper as well as connecting to the database and closing the connection after the test. Most of this testing was done as part of the submission for Phase 3.

```
private MessageDatabaseHelper dbHelper;



@BeforeEach

void setUp() throws SQLException {

   dbHelper = new MessageDatabaseHelper();

   assertTrue(dbHelper.messageConnectToDatabase(), "Database connection should
be successful.");

}



@AfterEach

void tearDown() {

   dbHelper.closeConnection();

}
```

Inputs: Each function will be checked by giving it its relevant input and checking if it modified the SQL table correctly. For example, the “addMessage” function will be tested with an input of `(1, 0, "Test help message", "Test search history");`

Which corresponds to the fields in the SQL table messageType, userID, message, searchHistory.

Outputs: All of the tests ran successfully.



Assessment: Each JUnit test will be evaluated by checking if the message database has been successfully modified from the input.

Coverage: As this requirement is for the creation of the message database and its associated functions, all of them need to be tested. As such, there are 9 test cases.

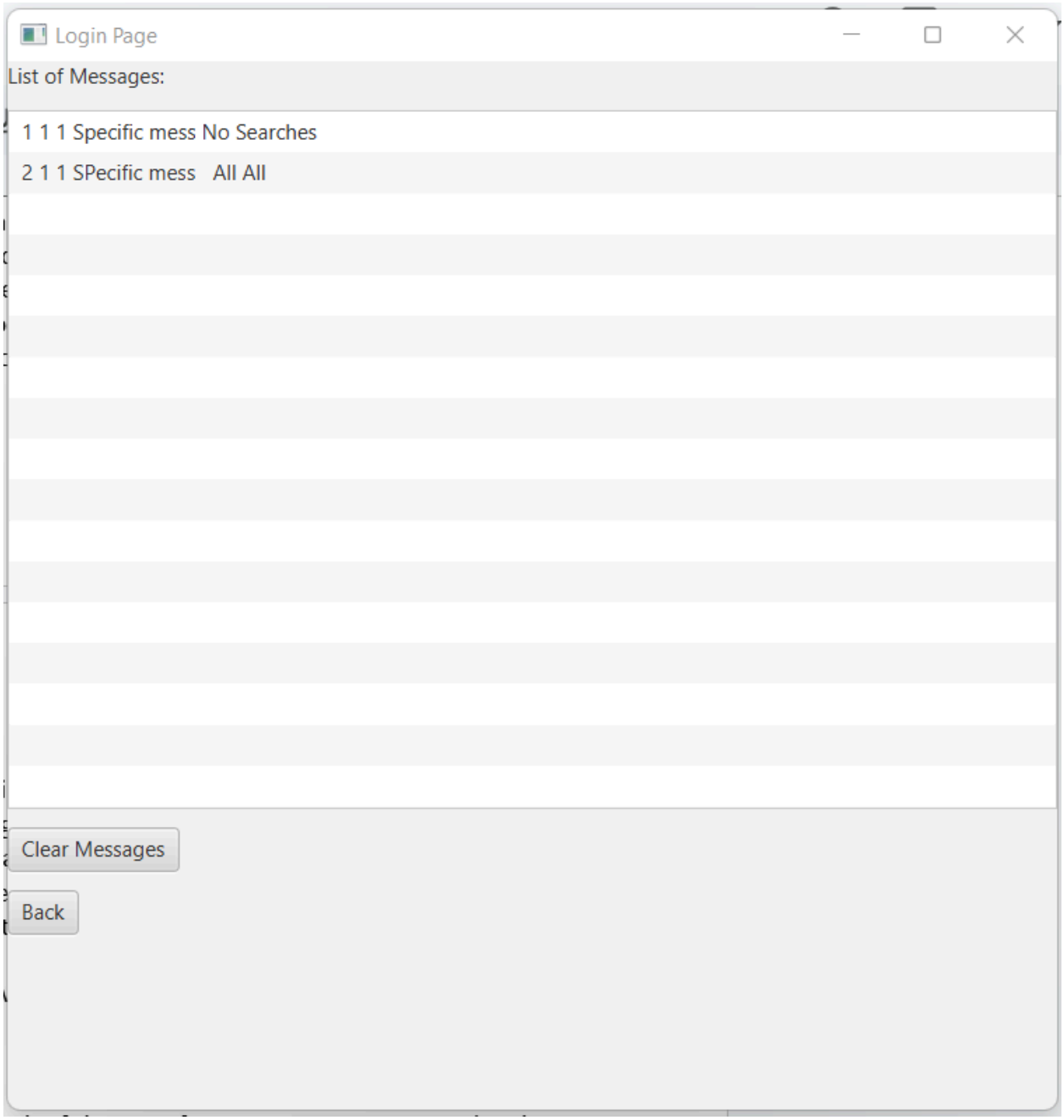
Requirement 3: The student role is enhanced with additional commands: Specific messages are those where the student cannot find the help information they need. In this last case, the student specifies what they need and cannot find. The system adds a list of the search requests the student has made. This information can be used to identify new help articles that need to be created and added to the system.

Setup: This test will be using the GUI, so the setup consists of logging in user1(who has student privileges) as a student.

Inputs: This test will be using the GUI, so the inputs consists of logging in user1 as a student (username = user1, password = a) then typing a help message in the text box (type in “Specific mess”), then clicking the “Send Specific Message” button. Additionally, the search history needs to be tested so a search will be performed by clicking the “Go to search articles” button then leaving all settings to default and performing a search. Click the “Back to Help Page” and proceed to send the exact same specific help message.

Outputs: The results will be examined by logging in as an admin and clicking the “View Messages” button. There should be two messages. The first should be “# 1 1 Specific mess No Searches” and the second “# 1 1 Specific mess a All All All”. The # will be different across tests since it is a system wide unique identifier for help messages. It is auto incremented upon creation of new messages and will never repeat, even if messages are deleted.

Assessment: The output is recorded in the following picture:



Based off the results, the test was a success. Both of the specific messages were created with the correct values as well as the search history.

Coverage: This requirement needed 2 tests to determine if it was fulfilled. The first test is to see if it correctly creates a specific help message, and the second is to see if it correctly records the search history.

Code

- The project code, nicely formatted with internal documentation, can be found at the GitHub link: https://github.com/princegoyani/g12CSE360.git

Screencasts

The screencasts are included in the linked folder

PHASE 4 SCREENCASTS

1. **Technical Screencast**: Demonstrate the code flow from requirements to application.
2. **User Screencast**: Show how to use the application for admins and instructional team members.

Credit Page

Team Member Name	**Task:**
Prince Goyani	Managing Java Project, Database and UI Integration, JUnit Testing, JavaFX UI, Special Access Creation
Connor Moon	JavaFX UI, Creation of Message Database, Article Search
Caitlin Chartier	JavaFX UI, Article Search, UI Testing and Improving
Keenan Tait	Database Architecture, Article Database, JUnit Requirements List, Project PDF
Nishtha Kukreja	