Project: CSE 360 Help System Phase 3

Alan Lintemuth, William McLean, Max Neville, Tushar Sachan, Taj Yoshimura

CSE 360: Thursday Group 47

Phase One Project Overview

Everyone struggles with finding the help they need to excel in life; few struggle as intensely as undergraduate students. As a remedy, our team is building a help system specifically designed for Computer Science and Engineering 360 students at Arizona State University. This system will leverage questions and answers from previous years' Education Discussion boards to fill out a computer application to connect students with the information they need. Students will benefit from personalized settings and the ability to seamlessly change roles depending on what information they would like to access. Instructors will benefit from a system that allows them to manage and assist students effectively. The system will be coded in the well-known coding language, Java, utilizing JavaFX to add easy to use graphical features. Our phase one deliverables will include all the methods necessary to build out our help system in the next phases.

In this system we have specified 3 specific roles of users: admins, students, and instructors. The role of admin has the ability to perform certain actions that the other user types cannot (though the role admin may also be possessed at the same time as the student or instructor role). These specific admin abilities are to:

- Invite someone to join the application by giving a new user a one-time password
- Reset a user account
- Delete a user account
- List the user accounts showing each user with their associated names and role(s)
- Remove or add a role from a user
- Log out

This differs from the abilities we give to all other users, students or instructors, which are to:

- Create an account from an invitation given by an admin utilizing a one-time password given from an admin, allowing a user to create an account with its own unique password
- Select their role: Admin, Instructor, and/or Student as allowed by an admin
- Log out

A major focus within this phase has been the security of the system. In order to keep our program, secure we added multiple measures such as a system of one-time passwords with expiration dates that are used within the process of creating new accounts, and keeping passwords protected by storing them as a more secure data type. By creating an enterprise-level help application, our team will be investing in the success of future students at ASU. With the tools generated in phase one, we will create a robust platform to build all future deliverables. We aim to create a private, secure, efficient platform that anyone can leverage to help succeed in computer science.

Phase Two Project Overview:

In this project phase, we will leverage the previously written code to create a database of help articles to be accessed by the admins and instructors. These roles will be able to interact with these individual help articles. This phase entirely focuses on implementing the article database and its addition to the overall system, with plans for further refinement and utilization in the future. After this phase, users of our system will be able to add and save completed articles to the database; these articles will later be accessed by all help system users to facilitate greater learning.

The articles within the database will each contain a unique ID to identify them alongside a title, authors, a set of keywords, a body, a level indication, a short description, references, and groups. Admins and instructors in the database will have the following abilities when it comes to interaction with the article database:

- Adding, Updating, Viewing, and Removing an article
- Listing articles (by group or as a whole)
- Backup articles to a file of the user's specification (by group or as a whole)
- Restore articles from a file (by merging the restored file with the existing database or replacing it entirely)

These functions will allow admins and instructors to efficiently create and operate the database. Groups of articles can be created, backed up, and restored so that each class at ASU can have the most pertinent help for the semester. Admins and instructors having the ability to list all the groups will make it easy to select from these slates. An additional layer of security is added in the form of encrypting the body of each article. When backed up to a file, the contents of the article's body will be encrypted. The body is only readable when the admin/instructor chooses to list the articles. All of this new functionality will build out our platform, getting closer to our goal of an efficient help system.

Change Bars: Project Phase Three

Our project is nearing completion as we complete the penultimate stage of development. This stage focuses on security for both users and instructors in the system. Moving forward, we will ensure all passwords are encrypted to keep all user accounts secure. A new class and group of articles is being created with encrypted bodies to keep proprietary information secure as well. Instructors can add users to these groups to allow viewing of decrypted articles. The student class is being added to allow a whole new class of user access to the help system. Students will be able to:

- **Send help queries to system administrators**
- **Search for relevant articles to answer their class questions**
- **Organize these searches by difficulty level and group designation**
- **Ask for access to special groups with encrypted articles**

Instructors will also be gaining new functionality to keep their information secure and allow easy creation of class groupings. They will be able to:

- **Create general groups for anyone to view articles on the system**
- **Create specialized groups to keep information confidential**
- **Create, view, edit, or delete articles and groups of articles**
- **Add, view, and delete students from the help system and groups**

This new functionality will allow group and article creation for any college class, research unit, and/or curriculum team. The administrator class will also be refined, requiring them to be added explicitly to groups while preserving most of their other functions. These additions will create the framework for our final secure product. After a few additional refinements in the next phase, our help system will be operational for any school needing a safe, searchable database of help for students.

Phase 1 Requirements and User Stories:

We will be formatting our User Stories in the following manner: As a *role* I want to *action* so that *benefit*

As a *user,* I want to *get help with CSE 360* so that *I can succeed in Computer Science.*

As a *new user*, I want to *set up my account properly so* that *my information will be stored correctly.*

As a *user,* I want to *be assigned the proper role so* that *I can access the information that most pertain to my questions.*

As a *user,* I want to *be able to switch roles* so that *I can access the correct information for my session.*

As a *user,* I want to *have a secure password* so that *my credentials will not be stolen*.

As *a user,* I want to *easily view my homepage so* that *I can gain an understanding of the help system*.

As a *user,* I want to *be able to log out* so that I *can finish my session*

As an *administrator*, I want to *establish an account to manage the system database* so that *it will be properly operated.*

As an *administrator*, I want *the first person to use the system to get an admin account* so that *the system will always have an administrator*.

As an *administrator*, I want *to invite people to join my application* so that *they can use my help system*.

As an *administrator*, I want to *choose the role for people when I send an invitation out* so that *they can assume the proper role.*

As an *administrator*, I want *to reset user accounts* so that *I can fix issues with their accounts.*

As an *administrator*, I want *to delete user accounts* so that *the database can function efficiently*.

As an *administrator*, I want to *be able to see all of the user accounts* so that *I can see who is using the application*.

As an *administrator*, I want to *change the roles of people using the help system* so that *I can keep users organized*.

Distilling down these user stories into Phase One Requirements gives us the best opportunity to focus on the needs of all stakeholders. Creating multiple user roles (Admin, Student, and Instructor) with corresponding home pages will be required to facilitate help system use and management. Data will have to be stored and utilized to organize the system. The system requires an administrator who can invite users, manage accounts, manage the system, and ensure all information is handled securely. Lastly, each user will require some functionality on their homepage to begin navigating the help system. By including all these requirements in Phase One, we will create a solid base to build out our ASU Student Help System.

Phase Two Requirements and User Stories:

As an *administrator*/*instructor*, I want to *be able to create articles* so that *I can add articles to the database.*

As an *administrator*/*instructor*, I want to *be able to add relevant information (such as titles, descriptions, bodies, etc.) to articles* so that *I can create meaningful help articles for students and instructors.*

As an *administrator*, I want to *ensure each article has a unique entry* so that *there will be no duplicate articles.*

As an *administrator*/*instructor*, I want to *be able to update, view, and delete articles* so that *I can modify articles and keep them valid.*

As an *administrator*/*instructor*, I want to *be able to backup and restore the articles* so that *I can keep the correct database.*

As an *administrator*/*instructor*, I want to *be able to create a separate backup file* so that *I can save a backup file safely.*

As an *administrator*/*instructor*, I want to *be able to create and backup a group of articles* so that *I can easily organize the help topics.*

As an *administrator*/*instructor*, I want to *be able to list all of the articles and groups of articles* so that *I can easily organize the help topics.*

The user stories of this phase represent the addition of the help articles and their functionality. Creating new articles with the correct fields is important to instructors and students seeking help. The ability to group articles by class or subject will streamline connections between users. Backing up and restoring the database and groups will ensure minimal interruption. The functionality added in this phase gives us a solid foundation for an efficient help system based on articles uploaded by admins and instructors.

Change Bars: Project Phase Three

As a *student,* I want to *be able to send messages to administrators/instructors* so that *I can get help with the help system articles.*

As a *student,* I want to *search for help articles* so that *I can find articles to assist me in my assignments.*

As a *student,* I want to *narrow down the difficulty level of articles I am searching for* so that *my search will efficiently return the articles I need.*

As a *student,* I want to *specify the group of articles I am searching for* so that *my search will return the correct articles.*

As a *student,* I want to *search the articles in the group of articles I am searching for* so that *my search will return the correct articles.*

As a *student,* I want to *have the search articles displayed efficiently* so that *choosing the right article will be easy.*

As an *instructor,* I want to *be able to view who has special permissions for a group* so that *I will know who to reach out to for access.*

As an *instructor,* I want to *be able to view which students have special permissions for a group* so that *I will know who has access and who needs it.*

As an *instructor,* I want to *be able to search just like the students* so that *I will know which articles they have access to.*

As an *instructor,* I want to *be able to create, view, edit, and delete special group articles* so that *I can control access to proprietary information.*

As an *instructor,* I want to *be able to add students to special groups* so that *they can access select articles with permission.*

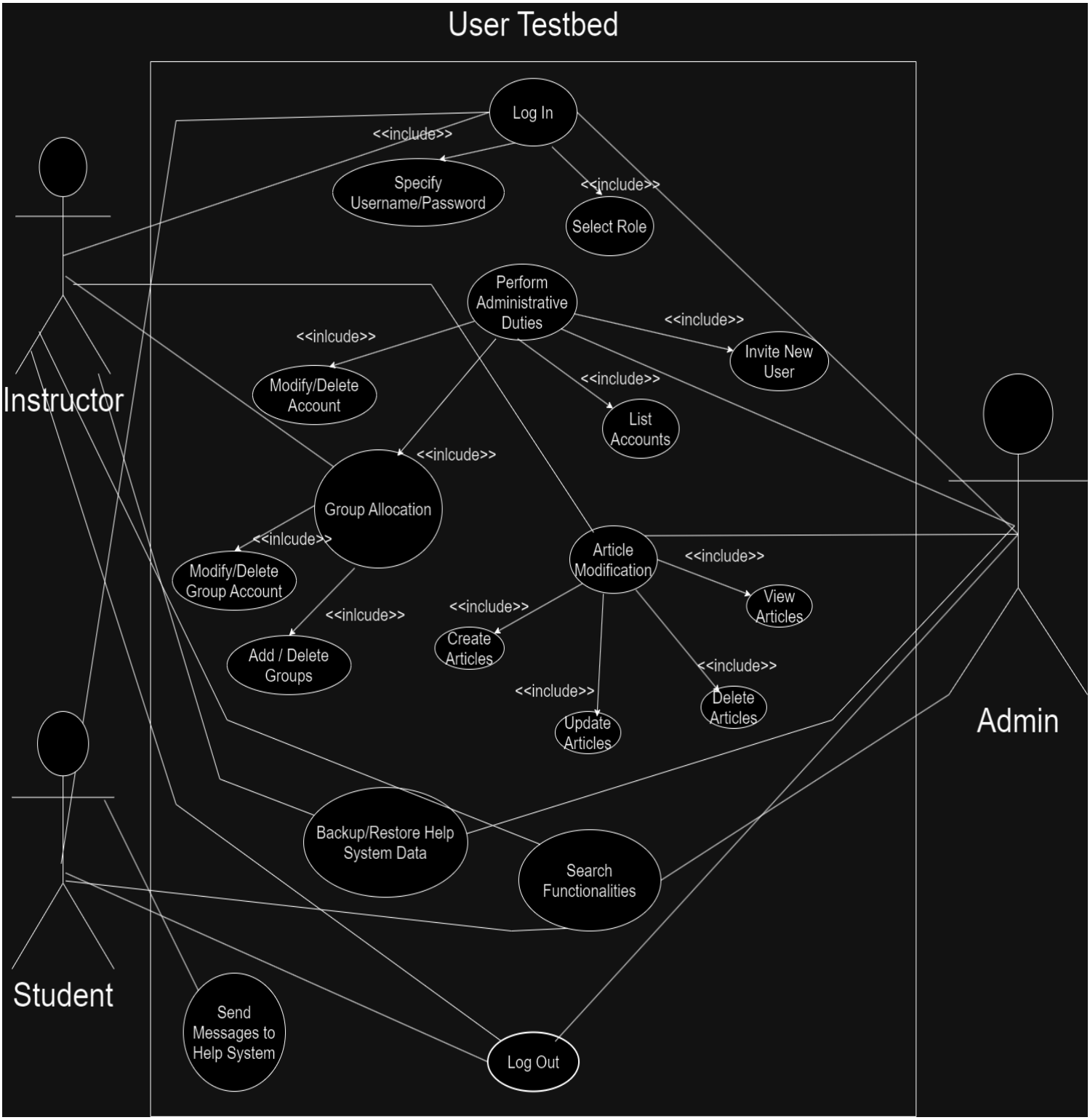
As an *instructor,* I want to *be able to remove users from special groups* so that *I can control who has access to the system.*

As an *administrator,* I want to *control general groups and which users are associated with them* so that *the system can be run efficiently.*

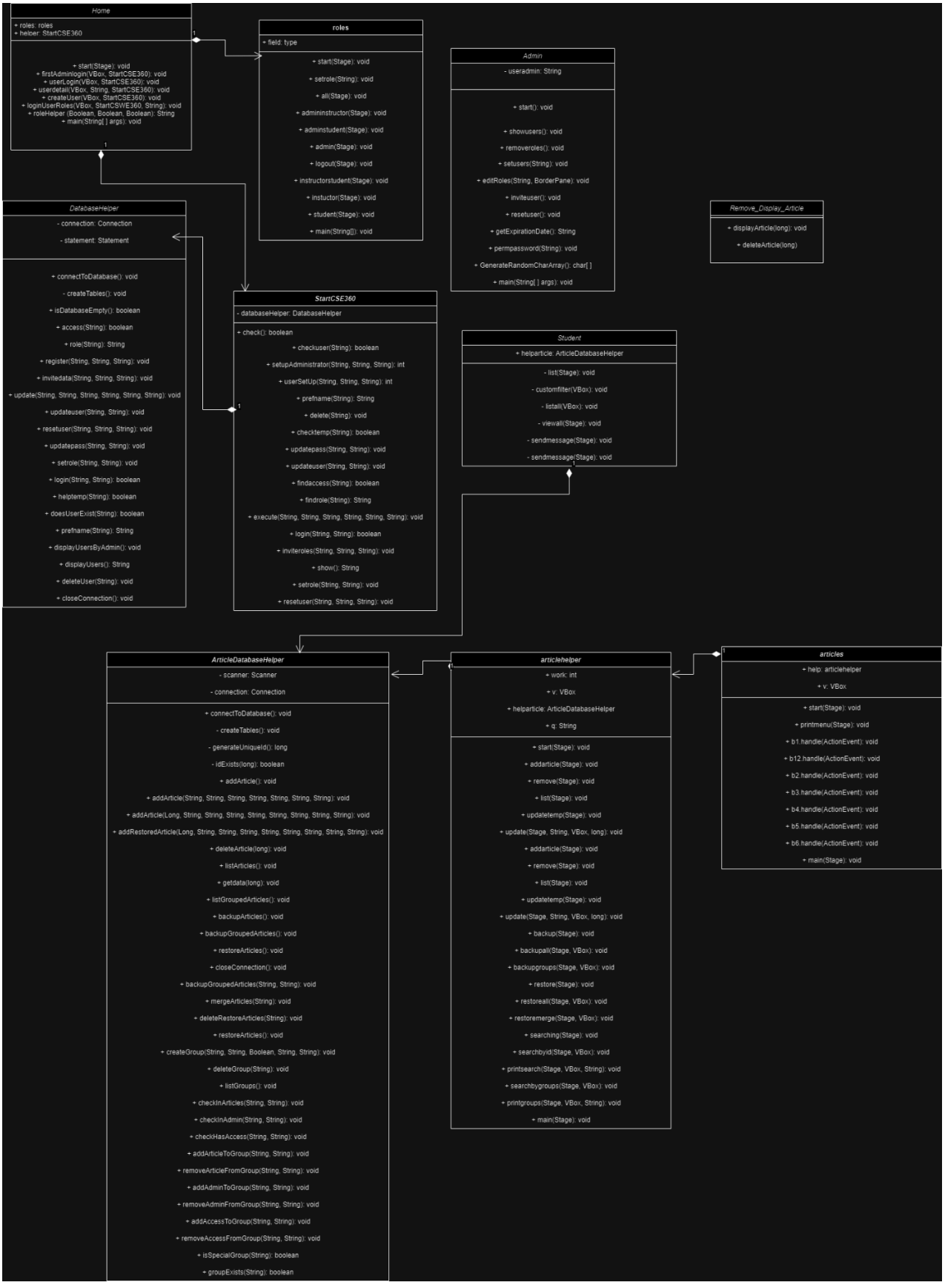
As an *administrator,* I want to *be able to receive special permission for a group* so that *I can help instructors manage their groups.*

This third phase creates protection for proprietary information within the help system. Instructors will be able to create encrypted articles and store them in special access groups. They can then add other users to their groups, ensuring specific special access articles can be viewed. Different instructors and administrators can be added to these special groups, while students, administrators, and instructors can be deleted. The student role is being created with the utility to send help messages, search for articles, view general articles, and view encrypted articles with special access. This section sets up our help system to become fully operational, with only one more phase of refinements.

Use Case:


User Testbed
Log In
<<include>>
Specify Username/Password
<<include>>
Select Role
Perform Administrative Duties
<<include>>
Invite New User
<<inlcude>>
Modify/Delete Account
<<include>>
List Accounts
Instructor
<<inlcude>>
Group Allocation
<<inlcude>>
Modify/Delete Group Account
<<inlcude>>
Add / Delete Groups
Article Modification
<<include>>
View Articles
<<include>>
Create Articles
<<include>>
Update Articles
<<include>>
Delete Articles
Admin
Backup/Restore Help System Data
Search Functionalities
Student
Send Messages to Help System
Log Out

UML:


Home
+ roles: roles
+ helper: StartCSE360
+ start(Stage): void
+ firstAdminlogin(VBox, StartCSE360): void
+ userLogin(VBox, StartCSE360): void
+ userdetail(VBox, String, StartCSE360): void
+ createUser(VBox, StartCSE360): void
+ loginUserRoles(VBox, StartCSWE360, String): void
+ roleHelper (Boolean, Boolean, Boolean): String
+ main(String[ ] args): void
roles
+ field: type
+ start(Stage): void
+ setrole(String): void
+ all(Stage): void
+ admininstructor(Stage): void
+ adminstudent(Stage): void
+ admin(Stage): void
+ logout(Stage): void
+ instructorstudent(Stage): void
+ instuctor(Stage): void
+ student(Stage): void
+ main(String[]): void
Admin
- useradmin: String
+ start(): void
+ showusers(): void
+ removeroles(): void
+ setusers(String): void
+ editRoles(String, BorderPane): void
+ inviteuser(): void
+ resetuser(): void
+ getExpirationDate(): String
+ permpassword(String): void
+ GenerateRandomCharArray(): char[ ]
+ main(String[ ] args): void
Remove_Display_Article
+ displayArticle(long): void
+ deleteArticle(long)
DatabaseHelper
- connection: Connection
- statement: Statement
+ connectToDatabase(): void
- createTables(): void
+ isDatabaseEmpty(): boolean
+ access(String): boolean
+ role(String): String
+ register(String, String, String): void
+ invitedata(String, String, String): void
+ update(String, String, String, String, String, String): void
+ updateuser(String, String): void
+ resetuser(String, String, String): void
+ updatepass(String, String): void
+ setrole(String, String): void
+ login(String, String): boolean
+ helptemp(String): boolean
+ doesUserExist(String): boolean
+ prefname(String): String
+ displayUsersByAdmin(): void
+ displayUsers(): String
+ deleteUser(String): void
+ closeConnection(): void
StartCSE360
- databaseHelper: DatabaseHelper
+ check(): boolean
+ checkuser(String): boolean
+ setupAdministrator(String, String, String): int
+ userSetUp(String, String, String): int
+ prefname(String): String
+ delete(String): void
+ checktemp(String): boolean
+ updatepass(String, String): void
+ updateuser(String, String): void
+ findaccess(String): boolean
+ findrole(String): String
+ execute(String, String, String, String, String, String): void
+ login(String, String): boolean
+ inviteroles(String, String, String): void
+ show(): String
+ setrole(String, String): void
+ resetuser(String, String, String): void
Student
+ helparticle: ArticleDatabaseHelper
- list(Stage): void
- customfilter(VBox): void
- listall(VBox): void
- viewall(Stage): void
- sendmessage(Stage): void
- sendmessage(Stage): void
ArticleDatabaseHelper
- scanner: Scanner
- connection: Connection
+ connectToDatabase(): void
- createTables(): void
- generateUniqueId(): long
- idExists(long): boolean
+ addArticle(): void
+ addArticle(String, String, String, String, String, String, String): void
+ addArticle(Long, String, String, String, String, String, String, String, String): void
+ addRestoredArticle(Long, String, String, String, String, String, String, String, String): void
+ deleteArticle(long): void
+ listArticles(): void
+ getdata(long): void
+ listGroupedArticles(): void
+ backupArticles(): void
+ backupGroupedArticles(): void
+ restoreArticles(): void
+ closeConnection(): void
+ backupGroupedArticles(String, String): void
+ mergeArticles(String): void
+ deleteRestoreArticles(String): void
+ restoreArticles(): void
+ createGroup(String, String, Boolean, String, String): void
+ deleteGroup(String): void
+ listGroups(): void
+ checkInArticles(String, String): void
+ checkInAdmin(String, String): void
+ checkHasAccess(String, String): void
+ addArticleToGroup(String, String): void
+ removeArticleFromGroup(String, String): void
+ addAdminToGroup(String, String): void
+ removeAdminFromGroup(String, String): void
+ addAccessToGroup(String, String): void
+ removeAccessFromGroup(String, String): void
+ isSpecialGroup(String): boolean
+ groupExists(String): boolean
articlehelper
+ work: int
+ v: VBox
+ helparticle: ArticleDatabaseHelper
+ q: String
+ start(Stage): void
+ addarticle(Stage): void
+ remove(Stage): void
+ list(Stage): void
+ updatetemp(Stage): void
+ update(Stage, String, VBox, long): void
+ addarticle(Stage): void
+ remove(Stage): void
+ list(Stage): void
+ updatetemp(Stage): void
+ update(Stage, String, VBox, long): void
+ backup(Stage): void
+ backupall(Stage, VBox): void
+ backupgroups(Stage, VBox): void
+ restore(Stage): void
+ restoreall(Stage, VBox): void
+ restoremerge(Stage, VBox): void
+ searching(Stage): void
+ searchbyid(Stage, VBox): void
+ printsearch(Stage, VBox, String): void
+ searchbygroups(Stage, VBox): void
+ printgroups(Stage, VBox, String): void
+ main(Stage): void
articles
+ help: articlehelper
+ v: VBox
+ start(Stage): void
+ printmenu(Stage): void
+ b1.handle(ActionEvent): void
+ b12.handle(ActionEvent): void
+ b2.handle(ActionEvent): void
+ b3.handle(ActionEvent): void
+ b4.handle(ActionEvent): void
+ b5.handle(ActionEvent): void
+ b6.handle(ActionEvent): void
+ main(Stage): void

Class Responsibility Collaborator:

CSE 360 Help System

Actors	Administrators, Instructors, Students, Help System, Help System Database
Description	An administrator will carry out all the functions of the database (inviting new users, resetting accounts, deleting accounts, changing user roles). Instructors and Students can sign up for an account, enter their information and password, and logout. Admins and Instructors can create, update, view, and delete help articles. Admins and Instructors can create and backup groups of articles, and list all the articles and subgroups.
User Data	Each actor's userID, password, first, middle, last, preferred name, and their role.
Article Data	Each article's header, title, authors, abstract, keywords, body of the article, group.
Stimulus	Invitation issued by an administrator, or a user logging into their account. Admin or Instructor adding and manipulating articles in the database; adding and manipulating groups of articles.
Response	System asks for a one-time login password, then accepts a new password. Conformation of logout. System asks for article data, group data, and backup/restore data.
Comments	Users must be specifically invited by an administrator. Their invitation comes with a one-time password that will expire. Two database tables are created to store user data and article data. Database tables can be backed up and restored from files.

Home

This JavaFX class is the entry point to our GUI and calls methods to login, sign up, and handle admin sign in

	Collaboration
Firstadminlogin(VBox, StartCSE360)	DatabaseHelper
usercreation(VBox, StartCSE360)	Roles
userlogin(VBox, StartCSE360)	
setrole(String, String): void	Roles
userdetail(VBox, , String, StartCSE360)	
String rolehelper(Boolean(B), B, B)	
loginUserRoles((VBox, StartCSE360, String)	

DatabaseHelper

This class deal with the data moving to and from the database

	Collaboration
role (String)	StartCSE360
register(String, String, String)	StartCSE360
update(String(S), S, S, S, S)	StartCSE360
displayUsers	StartCSE360
deleteuser(String)	StartCSE360
setUsers	StartCSE360

ArticleDatabaseHelper

This class deal with the data interfacing with the articles database

	Collaboration
connectToDatabase()	articlehelper
mergeArticles(string)	articlehelper
viewArticle(long)	articlehelper
addArticle(S, S, S, S, S, S, S, S)	articlehelper
addRestoredArticle(S, S, S, S, S, S, S, S)	articlehelper
deleteArticle(String)	articlehelper
listArticles()	articlehelper
listGroupedArticles()	articlehelper
backupArticles(String)	articlehelper
backupGroupedArticles(String, String)	articlehelper
addRestoredArticle()	articlehelper

Roles

This class handles the roles that the person has

	Collaboration
setrole(String)	Admin
administructor(Stage)	Instructor
adminstudent(Stage)	Student
admin(Stage)	Admin
logout(Stage)	Admin, Instructor, Student
instructor(Stage)	Instructor
student(Stage)	Student

Admin

This class handles the abilities of an admin

	Collaboration
Showusers	Roles
removeroles(User, enum)	Roles
Invite	CSE360
reset(User)	CSE360
delete(User)	CSE360
addRole(User, enum)	Roles
editroles(String, BorderPane)	
setUsers	

Users

This class handles the abilities of Users

	Collaboration
Showusers	Roles
removeroles(User, enum)	Roles
Invite	
reset(User)	
delete(User)	
addRole(User, enum)	Roles
editroles(String, BorderPane)	
setUsers	

articlehelper

This class is the front end for article database class and has functions to handle data

	Collaboration
addarticle(Stage)	ArticleDatabaseHelper
remove(Stage)	ArticleDatabaseHelper
list(Stage)	ArticleDatabaseHelper
update(Stage, String, VBox, long)	ArticleDatabaseHelper
backup(Stage)	ArticleDatabaseHelper
backupall(Stage, VBox)	ArticleDatabaseHelper
backupgroups(Stage, VBox)	ArticleDatabaseHelper
restore(Stage)	ArticleDatabaseHelper
restoreall(Stage, VBox)	ArticleDatabaseHelper
restoremerge(Stage, VBox)	ArticleDatabaseHelper
searching(Stage)	ArticleDatabaseHelper
searchbyid(Stage, VBox)	ArticleDatabaseHelper
searchbygroups(Stage, VBox)	ArticleDatabaseHelper
printgroups(Stage, VBox, String	ArticleDatabaseHelper

StartCSE360

This class is the front end for the database class and has necessary functions to handle data

	Collaboration
setupAdministartor(String, String, String): int	DatabaseHelper
userSetUp(String, String, String): int	DatabaseHelper
prefname(String): String	DatabaseHelper
delete(String): void	DatabaseHelper
findrole(String): String	DatabaseHelper
login(String, String): Boolean	DatabaseHelper

Screencasts:

Phase 3 Technical Screencast:

https://asu.zoom.us/rec/share/bZ1ZIaOXF-rd2RhP3mkBwLk-KnIPnswxsjmGmjvEyWQw4TMvo-NaBY5E2dobUZp-.QTXee3KHC0W0xOV_?startTime=1732165525000

Passcode: kN0!Ry.H

Phase 2 Technical Screencast:

https://asu.zoom.us/rec/share/tXCMb4Q2go_zI77TvtCsA_Nz7nt6YCSQU2NKuYZlmDz9Yq-CBytZyRZ5KugnYK6.mfhjFjAkhsVkHo8C

Passcode: Q2Z88OF^

Phase 1 Technical Screencast:

https://asu.zoom.us/rec/share/4xoL7wsDhevcYB-IWa99C2NkBXsQtZvDpQflO1E0WXQJ3YtOpHJim5KisIek5KHR.vHLLVy6AiRU9ZLNe?startTime=1728542604000

Passcode: P00Y%6r@

Phase 3 Demo Screencast:

https://asu.zoom.us/rec/share/9KinDaQN_lSwkPUTH85mveoh9_LRyBmi6B0_dvNqmVpu_29xyoxiobhk1OBGnlu8.q10GYN3NmvpZvZyC

Passcode: j+^.PE7J

Phase 2 Demo Screencast:

https://asu.zoom.us/rec/share/4AlwFvs8IUCLut2gvGcF__85y08RZYKTZQld5AYhNmd3L04VXLgrKvkKoH3UnZ2Y.yVdixQyWYT1B14_L?startTime=1730355374000

Passcode: q+xxbiP2

Phase 1 Demo Screencast:

https://asu.zoom.us/rec/share/m3yLcQi2a2pInwLi-mbtU-VGVfJ6OiiK5yhC1N9HBhvFVPR2Nu1bIY7p_2I4fSTI.hhEOIhqZ_Km8LJ4y?startTime=1728537570000

Passcode: n!FB03%X

GitHub Link:

https://github.com/mnevi/CSE360Project.git

Credit Page

Team Member Name	Contributions
Alan Lintemuth	Oversaw majority of the documentation including the Project Overview and User Stories segments. Worked with detailed testing of the project using JUnit functionality.
William McLean	Moral support.
Tuschar Sarchan	Implemented the entire GUI aspect for group functionality across the updated Student and Instructor classes; and helped with integration with the set functionality. Helped with testing/error analysis.
Max Neville	Completed much of the design and architecture for the project, creating the UML/Use Case diagrams. Also recorded both screencasts. Helped with testing / error analysis / credit page.
Taj Yoshimura	Worked on programming group functionality across the various classes, primarily working with the newly implemented group database. Also helped to create the CRC part of the document. Helped with testing / error analysis.

```java
package edu.asu.ASUHelloWorldJavaFXMaven;

import static org.mockito.Mockito.*;

public class JUnitTest {

    private DatabaseHelper dbHelper;
    private Connection mockConnection;
    private PreparedStatement mockPreparedStatement;
    private Statement mockStatement;


   @BeforeEach
    void setUp() throws SQLException {
        // You can use H2 in-memory database for testing
        String jdbcUrl = "jdbc:h2:mem:testdb"; // H2 in-memory DB
        mockConnection = DriverManager.getConnection(jdbcUrl, "sa", "");

    try (Statement stmt = mockConnection.createStatement()){

     String userTable = "CREATE TABLE IF NOT EXISTS cse360users ("
      + "id INT AUTO_INCREMENT PRIMARY KEY, "
      + "password VARCHAR(255), "
      + "role VARCHAR(20), "
      + "access BOOLEAN, "
      + "email VARCHAR(255), "
      + "first VARCHAR(255), "
      + "middle VARCHAR(255), "
      + "last VARCHAR(255), "
      + "preferred VARCHAR(255), "
      + "USERNAME VARCHAR(255), "
      + "temp VARCHAR(255), "
      + "date VARCHAR(255))";

     stmt.execute(userTable);
    }

        try (Statement stmt = mockConnection.createStatement()) {
            // Insert test users
            stmt.executeUpdate("INSERT INTO cse360users (password, role, access, email, first,
                    + "VALUES ('pass123', 'admin', true, 'admin', 'First', 'Middle', 'Last',
            stmt.executeUpdate("INSERT INTO cse360users (password, role, access, email, first,
                    + "VALUES ('pass456', 'user', false, 'user', 'SFirst', 'SMiddle', 'SLast',
        }

        dbHelper = new DatabaseHelper(mockConnection);
    }

/*    @Test
    void testIsDatabaseEmpty_whenDatabaseIsEmpty() throws SQLException {
        // When
        boolean isEmpty = dbHelper.isDatabaseEmpty();

        // Then
        assertTrue(isEmpty, "Database should be empty initially");
    }
*/
   @Test
   void testAccess() throws SQLException {
       // Given
       String username = "testUser";  // The username you're testing for
```

```
70          boolean expectedAccess = true; // The expected access value
71
72          // When
73          dbHelper.access(username); // Call the method to test
74
75          // Then
76          String query = "SELECT access FROM cse360users WHERE username = ?";
77          try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
78              pstmt.setString(1, username); // Set the username in the query
79
80              try (ResultSet rs = pstmt.executeQuery()) {
81                  assertTrue(rs.next(), "User should be found in the database");
82                  assertEquals(expectedAccess, rs.getBoolean("access"), "Access value should match
83              }
84          }
85      }
86
87
88      @Test
89      void testRegister() throws SQLException {
90          // Given
91          String username = "testUser";
92          String password = "testPassword";
93          String role = "student";  // Example role
94
95          // When
96          dbHelper.register(username, password, role); // Call the method to test
97
98          // Then
99          String query = "SELECT * FROM cse360users WHERE username = ? AND role = ?";
100         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
101             pstmt.setString(1, username); // Set the username in the query
102             pstmt.setString(2, role);     // Set the role in the query
103
104             try (ResultSet rs = pstmt.executeQuery()) {
105                 assertTrue(rs.next(), "User should be inserted into the database");
106                 assertEquals(username, rs.getString("username"));
107                 assertEquals(role, rs.getString("role"));
108                 // Optionally check the encrypted password:
109                 String encryptedPasswordFromDb = rs.getString("password");
110                 String expectedEncryptedPassword = Base64.getEncoder().encodeToString
111                 assertEquals(expectedEncryptedPassword, encryptedPasswordFromDb, "Passwords
112             }
113         }
114     }
115
116     @Test
117     void testInvitedata() throws SQLException {
118         // Given
119         String role = "role";
120         String temp = "temp123";
121         String date = "2024-11-20";
122
123         // When
124         dbHelper.invitedata(role, temp, date); // Call the method to test
125
126         // Then
127         String query = "SELECT * FROM cse360users WHERE role = ? AND temp = ? AND date = ?";
128         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
129             pstmt.setString(1, role);
130             pstmt.setString(2, temp);
```

```
            pstmt.setString(3, date);

            try (ResultSet rs = pstmt.executeQuery()) {
                assertTrue(rs.next(), "User should be inserted into the database");
                assertEquals(role, rs.getString("role"));
                assertEquals(temp, rs.getString("temp"));
                assertEquals(date, rs.getString("date"));
                }
        }
    }

    @Test
    void testUpdate() throws SQLException {
        // Given
        String username = "testUser";
        String initialEmail = "initialEmail@example.com";
        String initialFirst = "InitialFirst";
        String initialMiddle = "InitialMiddle";
        String initialLast = "InitialLast";
        String initialPreferred = "InitialPreferred";

        // Insert initial data into the database for the given username
        String insertUser = "INSERT INTO cse360users (username, email, first, middle, last,
        try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
            pstmt.setString(1, username);
            pstmt.setString(2, initialEmail);
            pstmt.setString(3, initialFirst);
            pstmt.setString(4, initialMiddle);
            pstmt.setString(5, initialLast);
            pstmt.setString(6, initialPreferred);
            pstmt.executeUpdate();
        }

        // New data to update
        String newEmail = "newEmail@example.com";
        String newFirst = "NewFirst";
        String newMiddle = "NewMiddle";
        String newLast = "NewLast";
        String newPreferred = "NewPreferred";

        // When: Call the method to update the user details
        dbHelper.update(newEmail, newFirst, newMiddle, newLast, newPreferred, username);

        // Then: Check if the data is updated in the database
        String query = "SELECT * FROM cse360users WHERE username = ?";
        try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
            pstmt.setString(1, username);

            try (ResultSet rs = pstmt.executeQuery()) {
                assertTrue(rs.next(), "User should be updated in the database");
                // Check updated fields
                assertEquals(newEmail, rs.getString("email"));
                assertEquals(newFirst, rs.getString("first"));
                assertEquals(newMiddle, rs.getString("middle"));
                assertEquals(newLast, rs.getString("last"));
                assertEquals(newPreferred, rs.getString("preferred"));
                // Check if access is set to true
                assertTrue(rs.getBoolean("access"));
            }
        }
    }
```

```
192
193     @Test
194     void testResetUser() throws SQLException {
195         // Given: Setup a user with a specific username and temp value to test
196         String username = "testUser";
197         String temp = "temp123"; // This is the temp value we will use to find the user
198         String date = "2024-11-20";  // The new date value to update
199
200         // Insert the user with a specific username and temp value
201         String insertUser = "INSERT INTO cse360users (username, temp, password) VALUES (?, ?,
202         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
203             pstmt.setString(1, username);
204             pstmt.setString(2, temp);
205             pstmt.setString(3, "oldPassword");  // Set an initial password
206             pstmt.executeUpdate();
207         }
208
209         // Confirm the user is inserted
210         String checkInsertQuery = "SELECT username, temp, password FROM cse360users WHERE
211         try (PreparedStatement pstmt = mockConnection.prepareStatement(checkInsertQuery)) {
212             pstmt.setString(1, username);
213             try (ResultSet rs = pstmt.executeQuery()) {
214                 assertTrue(rs.next(), "User should be inserted into the database");
215                 assertEquals(temp, rs.getString("temp"));
216                 assertNotNull(rs.getString("password"), "Password should not be null
217             }
218         }
219
220         // When: Call the resetuser method to reset the password, update the temp and date
221         dbHelper.resetuser(username, temp, date);
222
223         // Then: Check that the password is null, the temp is updated, and the date is set
224         String query = "SELECT password, temp, date FROM cse360users WHERE username = ?";
225         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
226             pstmt.setString(1, username);
227             try (ResultSet rs = pstmt.executeQuery()) {
228                 assertTrue(rs.next(), "User should be found in the database with the specified
229                 // Check that the password is set to null
230                 assertNull(rs.getString("password"), "Password should be set to null");
231                 // Check that the temp is still the same
232                 assertEquals(temp, rs.getString("temp"), "Temp value should remain the same");
233                 // Check that the date is set correctly
234                 assertEquals(date, rs.getString("date"), "Date should be updated");
235             }
236         }
237     }
238
239     @Test
240     void testUpdatePass() throws SQLException {
241         // Given: Setup a user with a specific username and password to test
242         String username = "testUser";
243         String oldPassword = "oldPassword123"; // Old password to insert
244         String newPassword = "newPassword456";  // The new password to update
245         String encryptedOldPassword = Base64.getEncoder().encodeToString(oldPassword.getBytes
246         String encryptedNewPassword = Base64.getEncoder().encodeToString(newPassword.getBytes
247
248         // Insert the user with an initial password
249         String insertUser = "INSERT INTO cse360users (username, password) VALUES (?, ?)";
250         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
251             pstmt.setString(1, username);
252             pstmt.setString(2, encryptedOldPassword);  // Insert the old password
```

```
253             pstmt.executeUpdate();
254         }
255
256         // Confirm the user is inserted and the password is correct
257         String checkInsertQuery = "SELECT username, password FROM cse360users WHERE username =
258         try (PreparedStatement pstmt = mockConnection.prepareStatement(checkInsertQuery)) {
259             pstmt.setString(1, username);
260             try (ResultSet rs = pstmt.executeQuery()) {
261                 assertTrue(rs.next(), "User should be inserted into the database");
262                 assertEquals(encryptedOldPassword, rs.getString("password"), "Password should
263             }
264         }
265
266         // When: Call the updatepass method to update the password
267         dbHelper.updatepass(newPassword, username); // Update the password
268
269         // Then: Check that the password is updated to the new encrypted password
270         String query = "SELECT password FROM cse360users WHERE username = ?";
271         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
272             pstmt.setString(1, username);
273             try (ResultSet rs = pstmt.executeQuery()) {
274                 assertTrue(rs.next(), "User should be found in the database with the specified
275                 // Check that the password is updated to the new encrypted password
276                 assertEquals(encryptedNewPassword, rs.getString("password"), "Password should be
277             }
278         }
279     }
280
281     @Test
282     void testSetRole() throws SQLException {
283         // Given: Create a user with an initial role
284         String username = "testuser";
285         String initialRole = "student";
286         dbHelper.register(username, "password", initialRole); // Assuming you have a register
287
288         // When: Update the user's role
289         String newRole = "admin";
290         dbHelper.setrole(newRole, username);
291
292         // Then: Verify the role is updated in the database
293         String query = "SELECT role FROM cse360users WHERE username = ?";
294         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
295             pstmt.setString(1, username);
296             try (ResultSet rs = pstmt.executeQuery()) {
297                 assertTrue(rs.next(), "User should be found in the database");
298                 assertEquals(newRole, rs.getString("role"), "Role should be updated to the new
299             }
300         }
301     }
302
303     @Test
304     void testLogin() throws SQLException {
305         // Given: Create a user with known credentials
306         String username = "testuser";
307         String password = "password123";  // This will be encrypted in the login method
308         dbHelper.register(username, password, "student"); // Assuming register method is
309
310         // When: Attempt to log in with correct credentials
311         boolean loginSuccessful = dbHelper.login(username, password);
312
313         // Then: The login should be successful
```

```
314         assertTrue(loginSuccessful, "Login should be successful with correct username and
    password");
315     }
316
317     void testHelpTemp() throws SQLException {
318         // Given: Insert a user with a known temp value
319         String temp = "temp123";
320         String username = "testuser";
321
322         dbHelper.register(username, "password123", "student"); // Assuming the register method
323
324         // Insert a row with the known temp value (directly using SQL for this test)
325         String insertTemp = "INSERT INTO cse360users (username, temp) VALUES (?, ?)";
326         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertTemp)) {
327             pstmt.setString(1, username);
328             pstmt.setString(2, temp);
329             pstmt.executeUpdate();
330         }
331
332         // When: Check if the temp value exists
333         boolean result = dbHelper.helptemp(temp);
334
335         // Then: The result should be true since the temp exists in the database
336         assertTrue(result, "Temp should exist in the database");
337     }
338
339     @Test
340     void testDoesUserExistWithExistingUser() throws SQLException {
341         // Given: A user "testuser" exists in the database
342
343         String username = "testuser"; // Known user
344
345         // When: Check if the user exists
346         boolean result = dbHelper.doesUserExist(username);
347
348         // Then: The result should be true since the user exists
349         assertTrue(result, "User should exist in the database");
350     }
351
352     @Test
353     void testDoesUserExistWithNonExistingUser() throws SQLException {
354         // Given: A user "nonexistentuser" does not exist in the database
355
356         String username = "nonexistentuser"; // Non-existing user
357
358         // When: Check if the user exists
359         boolean result = dbHelper.doesUserExist(username);
360
361         // Then: The result should be false since the user does not exist
362         assertFalse(result, "User should not exist in the database");
363     }
364
365
366     @Test
367     void testPrefnameWithExistingUserAndPreferredName() throws SQLException {
368         // Given: A user "testuser" with a preferred name "Pref"
369         String username = "testuser";
370
371         // When: Call the prefname method
372         String result = dbHelper.prefname(username);
373
```

```
374         // Then: The result should be the preferred name "Pref"
375         assertEquals("Pref", result, "The preferred name should be 'Pref'");
376     }
377
378     @Test
379     void testDisplayUsersByAdmin() throws SQLException {
380         // Given: Two users are inserted into the database.
381
382         // When: The displayUsersByAdmin method is called
383         dbHelper.displayUsersByAdmin();
384
385         // Then: Verify that the data for these users exists in the database.
386         String query = "SELECT * FROM cse360users WHERE id = ?";
387
388         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
389             pstmt.setInt(1, 1);  // Check for first user (ID = 1)
390             try (ResultSet rs = pstmt.executeQuery()) {
391                 assertTrue(rs.next(), "First user should be present in the database");
392                 assertEquals("test1@example.com", rs.getString("email"));
393                 assertEquals("password1", rs.getString("password"));
394                 assertEquals("admin", rs.getString("role"));
395             }
396
397             pstmt.setInt(1, 2);  // Check for second user (ID = 2)
398             try (ResultSet rs = pstmt.executeQuery()) {
399                 assertTrue(rs.next(), "Second user should be present in the database");
400                 assertEquals("test2@example.com", rs.getString("email"));
401                 assertEquals("password2", rs.getString("password"));
402                 assertEquals("user", rs.getString("role"));
403             }
404         }
405     }
406
407
408 }
```