Course: CSE 360 - Introduction to Software Engineering

Instructor: Professor Lynn Robert Carter, Ph.D.

Due Date: December 6, 2024

CSE 360 Help System Project
Phase Four Deliverable

Team Members:

Allan Binu

Jonathan Lin

Felix Allison

Ben Nelson

Sreehari Sreedev

Krystian Majchrzak

Change bars are formatted as follows: *like this for Phase 1*, *like this for Phase 2*, *like this for Phase 3*.

Project Overview:

The CSE 360 Help System is a secure, role-based application that manages user accounts and stores help articles. It allows admins and instructors to create, update, and organize articles, while students can easily search, access authorized articles, and submit questions. All of this is accomplished with a simple user interface and data that is encrypted and securely stored.

The CSE 360 Help System is a user management and information storage application designed to support secure, role-based access to information for admins, instructors, and students. This deliverable adds functionality for students to search for articles, access authorized articles, and submit questions. All article data will be encrypted and safely stored, with access only given to specific groups or users to safeguard sensitive information.

The CSE 360 Help System is a user management application designed to support account management and role-based access control for staff instructors and students. This deliverable, Phase Two, focuses on implementing core functions for article management, user access, and article storage. Admins and Instructors can now create, update, and organize help articles; with the option to group, restore, and back-up articles for more secure and efficient data management.

The CSE 360 Help System is a user management application designed to facilitate account creation, role-based access control, and administrative functions. This project represents the first phase of development, focusing on the core functions of logins, user management, and role assignment. The system is designed with multiple user role types in mind, admin and user roles (i.e. student and instructor), which have distinct permissions and abilities.

Requirements:

- All previous requirements remain persistent, unless otherwise stated

Improvements:

- The three most significant improvements of this deliverable were the database optimization, GUI overhaul, and general code polishing. For code polishing, any bugs discovered while recording the screencasts were dealt with and comments were also simplified to match a single standard. The GUI now looks much more professional due to buttons and textboxes being aligned to one another.

Feedback:

- For the first two phases, we had missed points on our automated testing. Both times, we had created some test files but we did not point them out with explanatory text, nor images. After adding these, we got full points for our Phase 3 Deliverable.

Articles:

- Searching for words or phrases in the body of the help articles

- *Search: (may only be done by students and instructors)*
  - *Can be performed using words, names, phrases in the title, author, abstract, ID number, or content level (e.g., beginner, intermediate, advanced, expert).*
  - *Can be filtered by a general group or special access group.*
  - *During a search, 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, the title, the author(s), and the abstract.*
  - *After a search, the user may perform:*
    - *another search*
    - *a request to view an entire article in detail using the sequence number,*
    - *one of the actions that are always available.*
  - *After viewing an article, the student can perform:*
    - *another search,*
    - *a request to view an entire article in detail using the sequence number,*
    - *one of the actions that are always available.*
- *Search Logs: (a list of the search requests the student has made)*
  - *Log Entries: shall store the search parameters (keywords, content level, group) every time a student performs a search.*
  - *Log Data Storage: may be stored as part of each student's profile data in the database, or in a centralized "Search Logs" table that links each search entry with the student's ID.*
  - *Log Duration: logs could be kept indefinitely for each student.*
- *Articles are created with the following data:*
  - *a title*
  - *the body of the help article*
  - *a set of links to reference materials and related articles*
  - *unique ID number, to detect duplicates when restoring*
  - *a unique header, includes information such as the level of the article (e.g., beginner, intermediate, advanced, expert)*
  - *grouping identifiers, so it is easy for the instructional team to update or delete a related set of articles*
  - *a short description, like an abstract for a paper, but shorter; similar to the short text provided by a web search*
  - *a set of keywords or phrases, to facilitate the search process for students*
- *Articles can only be backed up as part of a group.*
- *Articles may belong to more than one group.*
- *Other fields may be required to make it possible to find sensitive information and allow the easy grouping of articles.*

Groups:

General Groups:

- *Student:*
  - *Search and view articles (if given access).*
- *Instructor:*
  - *Search, view, create, edit, and delete articles.*
  - *Add, view, and delete students from the assigned general group.*
- *Admin:*
  - *Create and delete articles but cannot search, view, or edit any article body.*
  - *Add, view, and delete students and instructors from the assigned general group.*

Special Access Groups:

- *Student:*
  - *Search and view articles (if given access).*
- *Instructor:*
  - *Search, view, create, edit, and delete articles (if given access).*
  - *With special group rights: Add, view, and delete students and instructors from the special group.*
- *Admin:*
  - *Create and delete articles (if given access).*
  - *With special group rights: Add, view, and delete students, instructors, and admins from the assigned special group.*
- *A list of articles in the group where the body of the article is encrypted and decrypted*
- *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.)*
- *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.*
- *The default rights for new instructors added to this group do not include admin rights for this group.*
- *A list of instructors who have been given rights to view the decrypted bodies of articles in this group.*
- *A list of instructors given admin rights for this group.*
- *A list of students given viewing rights to the decrypted bodies of articles in the group.*
- *There must always be at least one user with admin rights for the special access group.*

Roles:

Login Functionality:

- *User's specified password is hashed and compared to the stored hash value*
- *Initial Login/Account Creation*
  - *First user is assumed to be an admin*
  - *User shall specify a username and password (password must be entered twice upon creation)*
    - *user's password is hashed and stored as the password value*
  - *The user is directed back to the original login page (has to log back in)*
- *Second Login (further account setup)*
  - *User is taken to a "Finish setting up your account" page and must enter:*
    - *email address*
    - *full name (first, middle, last, an optional preferred name)*
      - *If a preferred name is given, it should be used when displaying messages to that user*

Role Functionality:

Both admins and instructors shall be able to:

- *Add, view, update, remove any help article.*
- *List all the help articles and subsets of the help articles in one or more groups.*
- *Search and display articles that fit the user's search request.*
- *Backup grouped articles to the file.*
- *Restore grouped articles from the file.*
  - *If two ID numbers match, the backed-up copy will not be merged.*
  - *Users may remove all existing articles or merge the backup with current articles.*
- *Number of Roles per user*
  - *If user has multiple roles, they have to choose which role they wish to use for the session*
  - *If user has just one role, user is taken to the home page of that role*
  - *For Phase 1, the Student and Instructor role home pages have only one option, and that is to log out.*
- *Admin Page Functionality*
  - *Invite a user to join the application*
    - *Create a one-time code that a user can use to create a new account*
      - *Admin must specify which role(s) this invited user is given*
    - *The standard login page allows the user to provide a username to begin logging in or a different input field where they enter the invitation code*
  - *Reset a user password*
    - *Create a one-time password*
      - *Must be used on the user's next login*
      - *Needs expiration date & time*
    - *User is sent to "set up a new password" page, if password is **not** expired*
    - *User is sent back to the login page, once the new password is set*
  - *Delete a user account*
    - *Send an "Are you sure?" message, must be answered with "Yes"*
  - *List all user accounts*
    - *Includes the user name, the individual's name, and a set of codes for the roles is displayed.*
  - *Add or remove a role from a user*
  - *Log out*
- *Instructor & Student Page Functionality*
  - *To create an account, user need to fill in one-time invitation code*
    - *Account creation page only accepts a username and password*
  - *Account is created with roles given in the one-time invitation*
  - *User is sent to "Finish setting up your account screen" page*
    - *Then sent to the login page*
  - *For the session, a user with:*
    - *1 role is sent to the home page for the role*
    - *2 roles may select from available role pages for the session*
  - *For Phase 1, the only option for non-admin roles is to logout*

Miscellaneous Permissions:

- *Student:*
  - *Button to quit the application.*
  - *Send a generic/specific message to the help system to express confusion or request help when unable to find information.*
- *Instructor:*
  - *Backup and restore articles and article groups.*
- *Admin:*
  - *Backup and restore articles, article groups, and special access groups of articles.*

User Stories:

Admin:

- As an admin, I want to have the ability to easily and quickly go through the user interface, so that I find what I am looking for in an efficient manner.
- *As an admin, I want to have the ability to create, delete, back up, and restore help articles so that I can manage the system efficiently without viewing or editing the body of any article.*
- *As an admin, I want to be able to add, view, and remove users from general groups to maintain user access and group organization while ensuring at least one admin always has admin rights.*
- *As an admin, I want to be given admin rights for general and special access article groups so that I can manage these groups while adhering to the restriction of not viewing the article bodies.*
- *As an admin, I want to back up and restore general article groups and special access article groups so that I can ensure data recovery options are available.*
- *As an admin, I want to ensure that no admin rights are removed if it would leave the system, general group, or particular access group without an admin, to maintain proper system administration.*
- *As an admin, I want the ability to create, view, and delete access rights for special access groups while adhering to the rule that I cannot view the body of articles in these groups.*
- *As an admin, I want to be able to add a new article with a title, body, links, unique ID, header, description, and keywords so that users may have access to the information.*
- *As an admin, I want the ability to view a list of all articles or groups of articles to implement changes across said articles.*
- *As an admin, I want to be able to update the body and info of existing articles so that the article remains accurate and up-to-date.*
- *As an admin, I want the ability to delete individual help articles so that the system remains clean and organized.*
- *As an admin, I want to be able to group together help articles so that they are easier to organize, change, and manage.*
- *As an admin, I want to back up groups of articles so that said articles can be stored and used in case that data recovery is necessary.*
- *As an admin, I want to be able to restore help articles that have been backed up to safely recover after a data loss, while removing duplicate articles during a restoration.*
- *As an Admin, I want to list all users so that I can see the active users and manage their accounts.*
- *As an Admin, I want to generate a one-time invite code so that I can invite new users and assign them specific roles.*
- *As an Admin, I want to reset a user's password so that the user can access their account when they forget their credentials.*
- *As an Admin, I want to delete a user's account so that I can remove users who no longer need access to the system.*
- *As an Admin, I want to modify a user's roles so that I can adjust their access permissions as necessary.*
- *As an Admin, I want to log in to my account so that I can access the admin functionality of the system.*
- *As an Admin, I want to log out so that I can end my session and secure the system.*

Instructor:

- As an instructor, I want to have the ability to easily and quickly go through the user interface, so that I find what I am looking for in an efficient manner.
- *As an instructor, I want to be able to create, view, edit, and delete help articles to keep the information current and relevant for users.*
- *As an instructor, I want to have the ability to create, view, edit, and delete general and special access article groups so that I can organize and manage articles effectively.*
- *As an instructor, I want to back up and restore articles and groups of articles to ensure that data is recoverable in case of data loss.*
- *As an instructor, I want to add, view, and delete students from the help system and general groups to maintain student access and group organization.*
- *As an instructor added as the first member of a special access group, I want to be given admin rights to that group so that I can manage it effectively.*
- *As an instructor, I want to specify content levels when searching for help articles so that I can tailor the search results based on students' needs.*
- *As an instructor, I want to view a short list of articles that match my search criteria, which includes the group currently active, number of articles at each content level, and article details.*
- *As an instructor, I want to view the details of a specific article from the search results and perform actions such as viewing another article or starting a new search for ease of navigation.*
- *As an instructor, I want to be able to add a new article with a title, body, links, unique ID, header, description, and keywords so that students may have access to the information.*
- *As an instructor, I want the ability to view a list of all articles or groups of articles to implement changes across said articles.*
- *As an instructor, I want to be able to update the body and info of existing articles so that the article remains accurate and up-to-date.*
- *As an instructor, I want the ability to delete individual help articles so that the system remains clean and organized for the students.*
- *As an instructor, I want to be able to group together help articles so that they are easier to organize, change, and manage.*
- *As an instructor, I want to back up groups of articles for data recovery.*
- *As an instructor, I want to be able to restore backed-up help articles for continued student use in the event of a data loss, while also removing duplicate articles during a restoration.*
- *As an Instructor, I want to create an account using an invite code so that I can set up my access to the system based on my role, inserting my first, middle, and last name–with an optional preferred name.*
- *As an Instructor, I want to recover my account in case I forget my password, so that I can regain access to my information by using a one-time passcode given to me by an admin user..*
- *As an Instructor, I want to log in to my account so that I can access the system based on my role by using my username and password.*
- *As an Instructor, I want to log out so that I can end my session and secure my account by clicking the "Logout" button on my home page.*

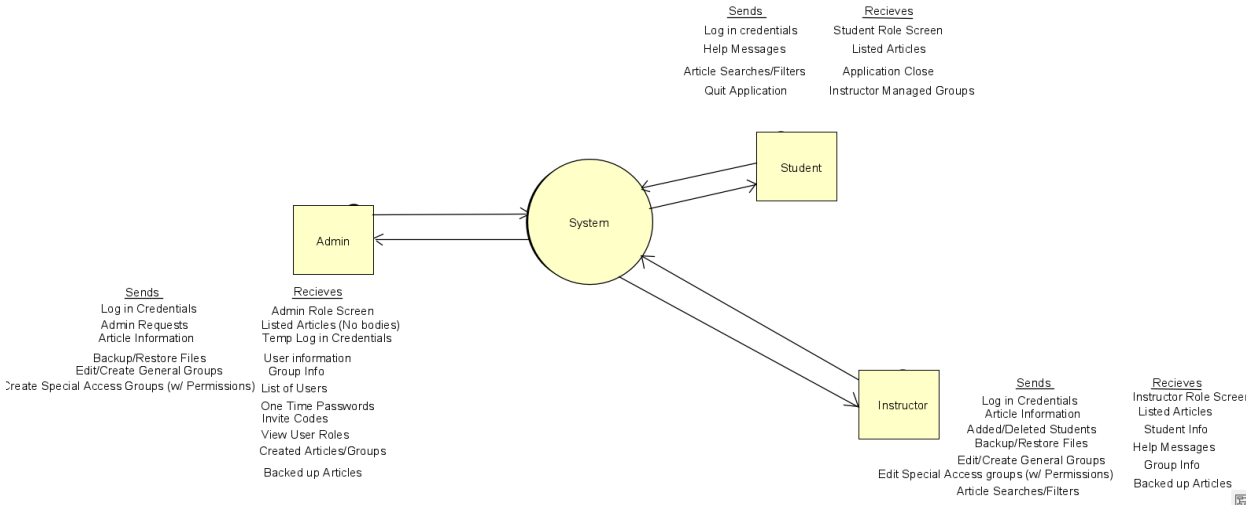
Student:

- As a student, I want to have the ability to easily and quickly go through the user interface, so that I find what I am looking for in an efficient manner.
- *As a student, I want to log in and log out of my account to access my role-based resources securely.*
- *As a student, I want to search for help articles using words, names, or phrases in the title, author, or abstract to find relevant information, without searching the body of the articles.*
- *As a student, I want to perform searches that can be limited to specific groups of articles to refine the results according to my needs.*
- *As a student, I want the search results to show the active group, the number of matching articles at each content level, and a short list with details*
- *As a student, I want to request to view an article in detail from the search results using its sequence number so that I can access more in-depth information.*
- *As a student, I want the ability to specify a content level or choose "all" to filter search results for articles at the desired difficulty*
- *As a student, I want the system to maintain a list of my search requests so that the admin or instructor can identify potential new help articles that need to be created.*
- *As a student, I want to have the right to view the bodies of articles only if I am given specific access rights, to maintain appropriate data security and privacy controls.*
- *As a Student, I want to create an account using an invite code so that I can set up my access to the system based on my role, inserting my first, middle, and last name–with an optional preferred name.*
- *As a Student, I want to recover my account in case I forget my password, so that I can regain access to my information by using a one-time passcode given to me by an admin user..*
- *As a Student, I want to log in to my account so that I can access the system based on my role by using my username and password.*
- *As a Student, I want to log out so that I can end my session and secure my account by clicking the "Logout" button on my home page.*

Architecture:

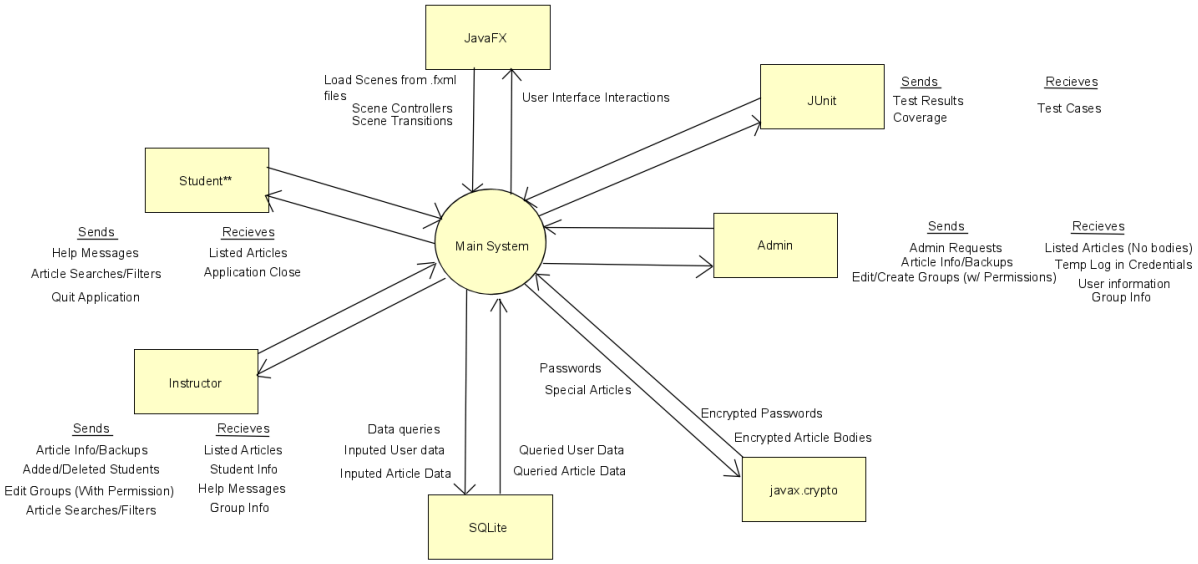
Context Diagram:

- The user inputs and outputs have been listed out to make them more readable.
- All three roles have been updated to take into account their new functionality.



Architectural Diagram:

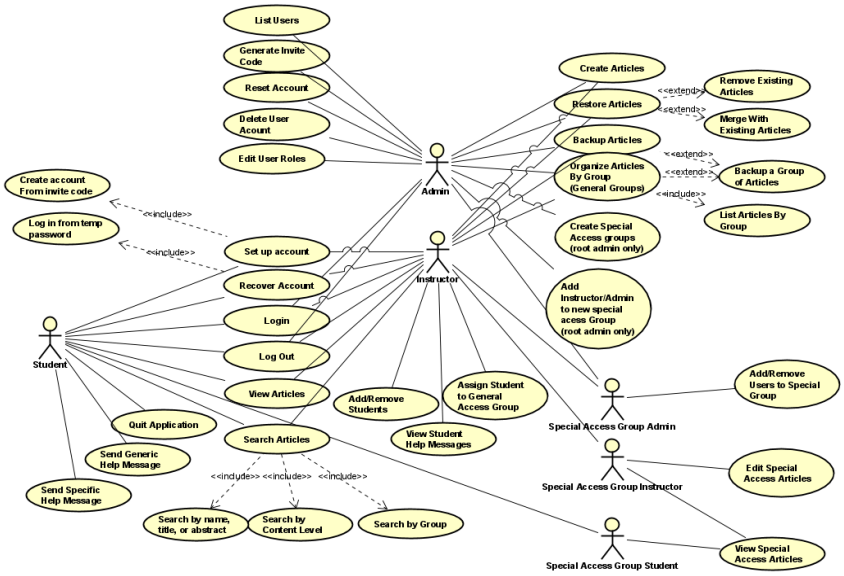
- The architecture diagram has been updated to list new user interactions with the system.
- The new addition to this phase is the introduction of JUnit testing. This helps with testing methods in classes and easily showing what the test cases cover.



Design:

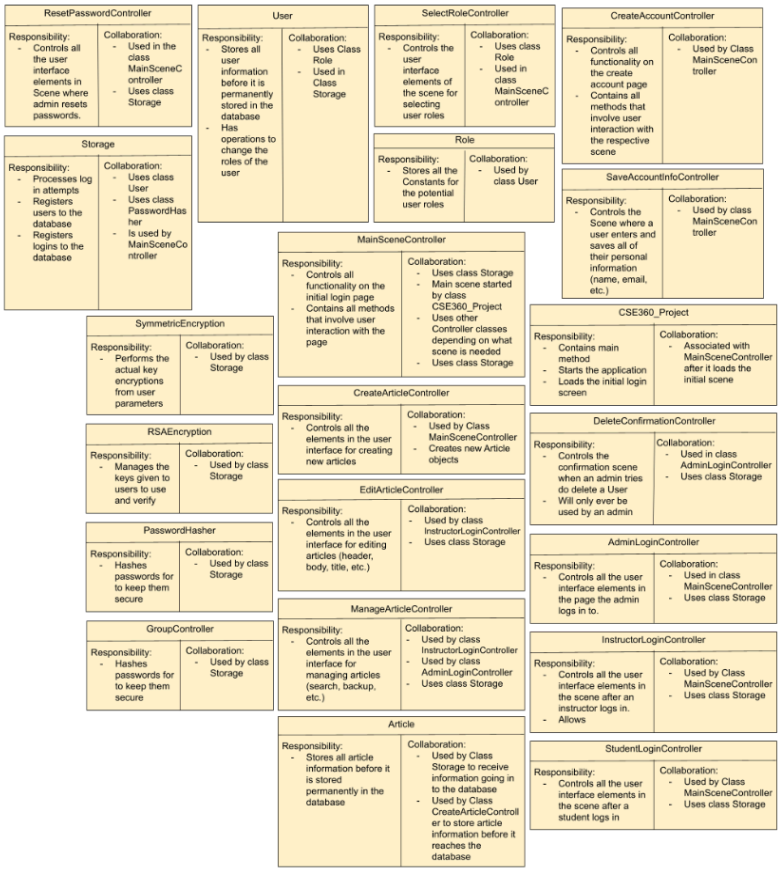
Use Case Diagram:

- Use cases along the left side display user actions, while new use cases along the right focus on article management and encryption. Search functionality has been added to the bottom-left.



Class Responsibility Collaborator Cards:

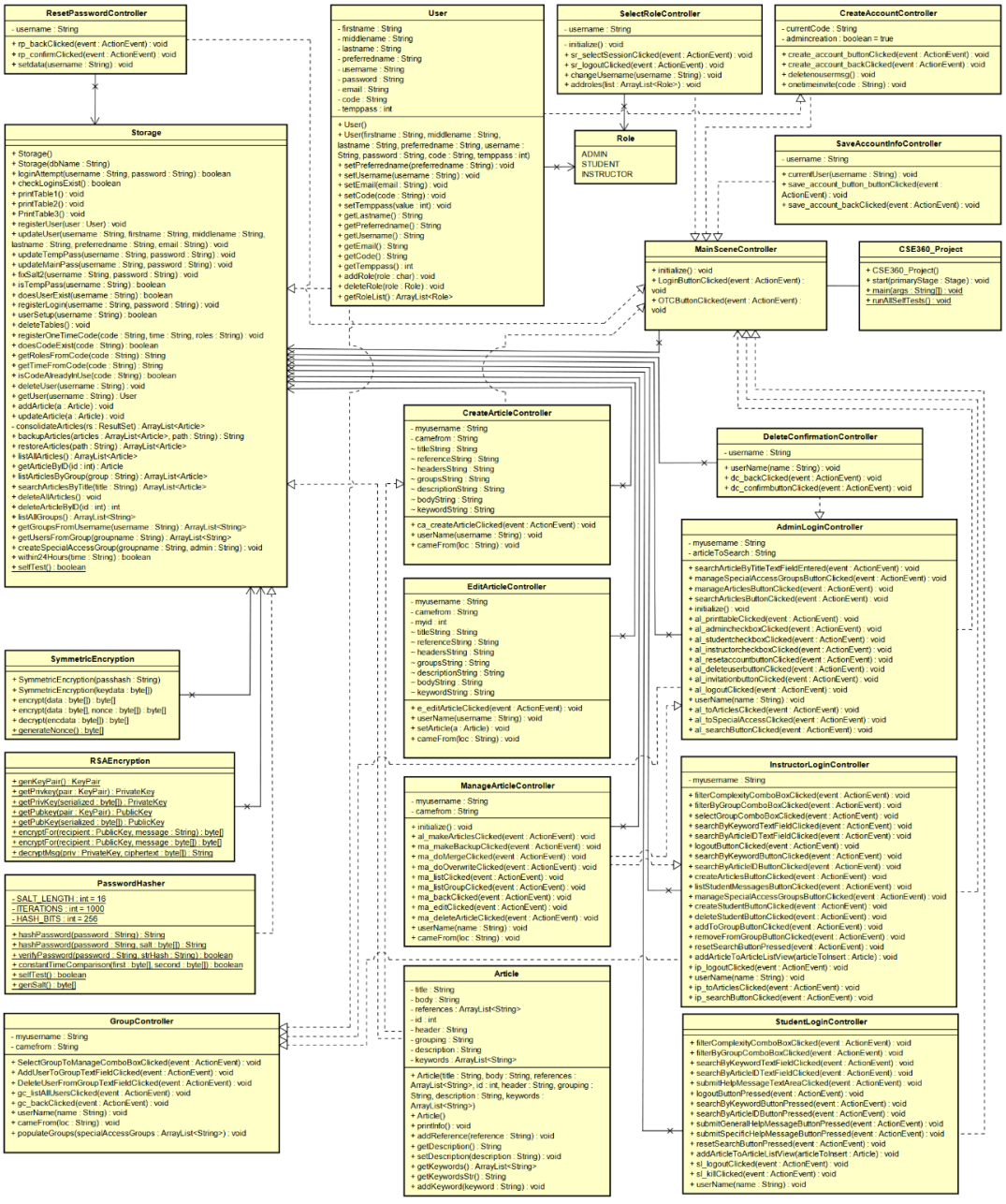
- Four new classes have been added:
  - EditArticleController, RSAEncryption, SymmetricEncryption, & GroupController
- EditArticleController is only accessible by instructors, if given the proper permissions.



Class Diagram:

Additions to class design in this phase include:

- GroupController allows users with admin rights to manage permissions for general and special access groups.
- RSAEncryption and SymmetricEncryption handle the encryption/decryption of articles.
- Expanded article management to support encryption and better access control with EditArticleController.
- Student users have increased functionality, adding search, filter, and help request features.



Testing:

All JUnit files end in `test.java`

- This section will provide an overview of internal testing for the system's internal methods
- JUnit was utilized to efficiently demonstrate automated test cases of the non-GUI classes in this project. Trivial method testing was left out of the doc, but below is a summary of some of our critical tests.
- The screenshot below shows the coverage summary of all created test cases.
    - Note: no controller or GUI classes receive JUnit test coverage.

src	39.8 %	4,044	6,127	10,171
(default package)	39.8 %	4,044	6,127	10,171
Article.java	100.0 %	229	0	229
Role.java	100.0 %	34	0	34
SymmetricEncryption.java	100.0 %	164	0	164
User.java	100.0 %	204	0	204
AllTest.java	98.6 %	2,039	29	2,068
PasswordHasher.java	97.8 %	133	3	136
RSAEncryption.java	85.9 %	67	11	78
Storage.java	68.8 %	1,174	532	1,706

- Info on GUI tests can be found in the screencasts.

User Creation Testing:

Tested Requirement:

- A user should be able to log in and be registered to the database.

Setup/Inputs:

- For this test, a test database is set up and two users are created with different role permissions. Both will be added to the database for the test.

Test:

```
82    @Test
83    void testRegisterUser() throws NoSuchAlgorithmException, InvalidKeySpecException, SQLException {
84        Storage s = new Storage("systmedb6");//Create the test database
85        User u = new User("Benjamin","D", "Nelson", "Ben", "ben123", "2345","34", 123); //Create a user to add
86        u.addRole('a'); //Add each role to the user
87        u.addRole('i');
88        u.addRole('s');
89        s.registerUser(u);
90
91        User m = new User("B","D", "Nel", "Ren", "n23", "2345","34", 123); //Create a second user
92        m.addRole('s');
93        s.registerUser(m);
94
95        s.printTable2(); //Print the user table to show results
96    }
97
```

Output/Results:

```
Console × JUnit Git Staging
<terminated> StorageTest (1) [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (Dec 6, 2024, 5:41:21 PM – 5:41:24 PM) [pid: 30072]
Could not find user tester!
----logins table contents----
----user_info table contents----
number of users in user_info: 0
----logins table contents----
----user_info table contents----
User            #1
Username:       ben123
First Name:     Benjamin
Middle Name:    D
Last Name:      Nelson
Preferred Name: Ben
Roles:          AIS
Invitation Code:        34
User            #2
Username:       n23
First Name:     B
Middle Name:    D
Last Name:      Nel
Preferred Name: Ren
Roles:          S
Invitation Code:        34
number of users in user_info: 2
Could not find user ben123!
User to delete does not exist
```

- After adding the users to the database, printing the user table shows that both users and their respective information has been successfully added. This indicates that the test was a success.

Tested Requirement:

- Admins should be able to specify the role when they create a user. This is a very important feature for security, so it's important each one time access code can identify what role the user that gets it receives.

Setup/Inputs:

- This test will set up two one time codes with different roles assigned to them. The getRolesFromCode method should be able to return the specified roles.

Test:

```
227     @Test
228     void testGetRolesFromCode() throws SQLException {
229         Storage s = new Storage("systmedb19"); //create database
230         s.registerOneTimeCode("1234", "765", "admin");  //create one time code
231         s.registerOneTimeCode("2234", "7765", "instructor");
232
233         System.out.println("Code 1234's Role: "+s.getRolesFromCode("1234"));//print roles from the code
234         System.out.println("Code 2234's Role: "+s.getRolesFromCode("2234"));
235     }
```

Output/Results:

```
Console × JUnit Git Staging
<terminated> StorageTest (1) [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (Dec 6, 2024, 5:51:43 PM – 5:51:47 PM) [pid: 8540]
Code 1234's Role: admin
Code 2234's Role: instructor
```

- The method is able to properly return each code's respective roles, indicating that this test is a success.

Encryption Testing:

Tested Requirement:

- Special Access groups should have the article bodies encrypted for safe keeping. We have used RSA encryption to do this

Setup/Inputs:

- To test the encryption methods, we have set up two tests, one checking encryption, and one testing decryption. Both take a string converted to a byte array as input. In the following tests, "hello there" is used as the test string.

Tests:

```
48    @Test
49    void testEncryptForPublicKeyByteArray() throws InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingExcep
50        byte[] message = "hello there".getBytes(); //message to be encrypted (as bytes)
51
52        byte[] cipher = RSAEncryption.encryptFor(RSAEncryption.getPubkey(RSAEncryption.genKeyPair()), message); //encrypt message
53
54        System.out.println("ciphertext: "); //prints encrypted bytes of message
55        for(int i=0;i<cipher.length;i++) {
56            System.out.print(cipher[i]);
57        }
58        System.out.println();
59
60    }
61
62    @Test
63    void testDecryptMsg() throws InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {
64
65        byte[] message = "hello there".getBytes(); // message to be encrypted
66        KeyPair key =RSAEncryption.genKeyPair(); //generate a key pair for encrypting and decrypting
67        byte[] ciphertext = RSAEncryption.encryptFor(RSAEncryption.getPubkey(key), message); //encrypt the message
68
69        String newtext = RSAEncryption.decryptMsg(RSAEncryption.getPrivkey(key), ciphertext); //decrypt the message
70        System.out.println("Decrypted text: " + newtext);
71    }
72
73 }
74
```

```
Console × JUnit Git Staging
<terminated> RSAEncryptionTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (Dec 6, 2024, 4:40:42 PM – 4:40:44 PM) [pid: 23532]
ciphertext:
90-78-114-127-742347-1107-44-63-111-88-92-114-5234991839106121-3-3844-7667-9750-10-4183-34383-78-614966-62-18-70-66-97-52-100-4-4888-83-102-30-32394-7217397-56-92-4189-105-1036930-853229-42102-8129948142-73-870659220-
Decrypted text: hello there
```

Output/Results:

- The Encryption method should output cipher text of the encrypted string, and the decrypt method should output the original string after being decrypted. Seeing the outputs in the console, we can see that this test was successful.

Testing Article/Group Functionality:

Tested Requirement:

- Admins/Instructors can create Articles.

Setup/Inputs:

- This method takes an article that has been created beforehand and adds it to the database. In this case, two articles are added, one with a valid id (article a) and one with an invalid (article b).

Test:

```
269
270    @Test
271    void testAddArticle() throws SQLException {
272
273        Storage s = new Storage("systmedb24"); //Create Test Database
274
275        ArrayList<String> references = new ArrayList<String>(); //Create Article to Add
276        ArrayList<String> keywords = new ArrayList<String>();
277        references.add("wiki");
278        keywords.add("help");
279        Article a = new Article ("Databases help", "fdgfsdg", references, 1, "Step 1", "db","helps you", keywords);
280        Article b = new Article ("Databases help", "fdgfsdg", references, 0, "Step 1", "db","helps you", keywords);
281
282        s.addArticle(a);//adds articles and prints results
283        s.addArticle(b);
284        s.listAllArticles();
285    }
```

Expected Output/Results:

```
589    public void addArticle(Article a) throws SQLException {
590        String update = "INSERT INTO articles (title, body, refs, id, header, grouping, description, keywords, isSecure) VALUES (?,?,?,?,?,?,?,?,?)";
591        PreparedStatement prep = conn.prepareStatement(update);
592        String references = a.getReferencesStr();
593        String keywords = a.getKeywordsStr();
594        int id = a.getID();
595
596        prep.setString(1, a.getTitle());
597        prep.setString(2, a.getBody());
598        prep.setString(3, references);
599        if (id > 0) {
600            prep.setInt(4, a.getID());
601        } else {
602            prep.setNull(4, Types.NULL);
603        }
604        prep.setString(5, a.getHeader());
605        prep.setString(6, a.getGrouping());
606        prep.setString(7, a.getDescription());
607        prep.setString(8, keywords);
608        prep.setInt(9, 0);
609
610        prep.executeUpdate();
611    }
```

- The addArticle method runs successfully and the test also manages to ensure coverage of the entire method, therefore this test is a success.

Tested Requirement:

- Instructors/Students can search for users by group id. This test will ensure that the internal search method is operating correctly.

Setup/Inputs:

- This test sets up a database with an article to be searched for, and then calls the getArticleById method for that article. Additionally, the method is called on an article/ID that does not exist to ensure that is handled correctly.

Test:

```
346
347    @Test
348    void testGetArticleByID() throws SQLException {
349        Storage s = new Storage("systmedb29");//Create a test database
350
351        ArrayList<String> references = new ArrayList<String>(); //Create an article for testing
352        ArrayList<String> keywords = new ArrayList<String>();
353        references.add("wiki");
354        keywords.add("help");
355        Article a = new Article ("Databases help", "fdgfsdg", references, 1, "Step 1", "db","helps you", keywords);
356        s.addArticle(a);
357
358        s.getArticleByID(1); //Search for an article that exists
359        s.getArticleByID(2); //Search for an article that does not exist
360    }
```

Output/Results:

```
932    public Article getArticleByID(int id) throws SQLException {
933        String query = "SELECT * FROM articles WHERE id = ? AND isSecure = 0";
934        PreparedStatement prep = this.conn.prepareStatement(query);
935        prep.setInt(1, id);
936        ResultSet rs = prep.executeQuery();
937
938        ArrayList<Article> got = consolidateArticles(rs);
939        if (got.size() < 1) {
940            return null;
941        }
942        return got.get(0);
943    }
944
```

- The test is able to successfully run through the getArticleByID method, including the case when the searched for article does not exist. The total coverage for this case means that this test is a success.

Tested Requirement:

- Admins can create special access groups.

Setup/Inputs:

- This test will ensure that when a special access group is created, the database is also successfully updated to reflect the new group. The test requires a database to be created with an admin user, which is pre-loaded in the database file. The createSpecialAccessGroup method will be called twice, once for a valid admin user, and once for an invalid one.

Test:

```
435  @Test
436  void testCreateSpecialAccessGroup() throws SQLException, InvalidKeyException, NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingExce
437
438      Storage s = new Storage(); //load storage
439      s.createSpecialAccessGroup("testGroup", "tester"); //create group with invalid admin
440      s.createSpecialAccessGroup("testSecretGroup", "admin");  //create group with valid admin
441  }
```

Output/Results:

```
1069*    public boolean createSpecialAccessGroup(String groupname, String admin) throws SQLException, NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException, NoSuchPaddingException, Ille
1070         // 1. Add the special_groups entry
1071         // 2. Generate an encryption key to be used for the group
1072         // 3. Encrypt the group key with RSA for the public key of the admin user
1073         // 4. store this special_access record
1074         byte[] group_key_raw = new byte[32];
1075         SecureRandom random = new SecureRandom();
1076         random.nextBytes(group_key_raw);
1077
1078         String query2 = "SELECT pubkey FROM logins WHERE l_user = ?";          //Get the pubkey from the user
1079         PreparedStatement prep2 = this.conn.prepareStatement(query2);
1080         prep2.setString(1, admin);
1081         ResultSet rs = prep2.executeQuery();
1082         byte[] pubkey_raw;
1083         if (!rs.next()) {
1084             System.out.println("Could not find user " + admin + "!");
1085             return false;
1086         }
1087         pubkey_raw = rs.getBytes("pubkey");                                   //Generate Encryption Key
1088         PublicKey pubkey = RSAEncryption.getPubKey(pubkey_raw);
1089         byte[] group_key_enc = RSAEncryption.encryptFor(pubkey, group_key_raw);
1090
1091         String query1 = "INSERT INTO special_groups (group_name) VALUES (?)"; //Add special Groups to database
1092         PreparedStatement stmt1 = this.conn.prepareStatement(query1);
1093         stmt1.setString(1, groupname);
1094         stmt1.executeUpdate();
1095
1096         String query15 = "SELECT last_insert_rowid() AS the_id";
1097         Statement stmt15 = this.conn.createStatement();
1098         ResultSet rs0 = stmt15.executeQuery(query15);
1099         if (!rs0.next()) {                                                    //Catch case: group creation fails
1100             System.out.println("Group creation failed!");
1101             return false;
1102         }
1103         int group_id = rs0.getInt("the_id");
1104
1105         String query3 = "INSERT INTO special_access (s_user, access_group_id, group_key) VALUES (?,?,?)";
1106         PreparedStatement prep3 = this.conn.prepareStatement(query3);
1107         prep3.setString(1, admin);
1108         prep3.setInt(2, group_id);
1109         prep3.setBytes(3, group_key_enc);
1110         prep3.executeUpdate();
1111         return true;
```

- The special access groups are successfully created and added to the database through the method. Each case of having a valid and invalid admin runs successfully. The only portion without coverage is a check in the case that it fails to add the group to the database. This means that this test is a success.

Screencasts:

Technical Screencast:

- https://drive.google.com/file/d/1vQ-1SAaGnqivCTzmLb38wkUiq2Vd1v0e/

How-To-Use Screencast:

- https://drive.google.com/file/d/1281zOYHqCX03w6BV6J7-_reYlz5PrHoK/

Outline:

- What we had decided to do was split each video into two teams. Felix Allison & Sreehari Sreedev for the technical screencast, and Allan Binu, Ben Nelson, & Krystian Majchrzak for the how-to-use screencast. The how-to-use screencast was split up into three parts for each presenter: user functionality, special groups & article management, and search & filtering.

Github Code:

Phase 4: https://github.com/sreehax/CSE360_Phase4

- Please read the bottom of README.md for the proper jar files.

Credit Page:

Member Names	**Contributions**
Allan Binu	Program Documentation How-To-Use Screencast
Jonathan Lin	Nothing
Felix Allison	UI Programming Technical Screencast
Ben Nelson	Testing Documentation How-To-Use Screencast
Sreehari Sreedev	Database Programming Technical Screencast
Krystian Majchrzak	Deliverable Documentation How-To-Use Screencast