CSE 360 Help System Project Phase 4

Christian Halas, Ryan Reny, Matthew Ager, Ethan Rudy, Kathan Parag Shah, Botirbek Omonov

Arizona State University

CSE 360 - Introduction to Software Engineering

Professor Lynn Robert Carter, Ph.D.

December 6, 2024

Project Overview:
In this phase (aka the final phase) some automated test code for the testing part of the assignment. We also worked on coding what was left to code based on the requirements. As for the feedback from the grader, we fixed the issues he mentioned that were wrong with the submission. The first issue is that we had no automated testing code, so we created some testing code and added it. The second issue is that we didn't create a UML diagram for the last phase so we are fixing that. The third issue is that we had created our wireframes by hand and not digitally, so we fixed that. The fourth issue is that we didn't make the stuff in the design section into tables, unfortunately we didn't get to this. 1 of our group unfortunately wasn't able to participate in the screencasts due to schedule problems and us finding out about needing to do the screencasts together late.

Requirements/User Stories:

Phase 1 user stories:

Admin User Stories

1. **Account Creation and Role Assignment**
   - *As the first user to log into the system,* I want to create an account with a username and password so that I am automatically assigned the Admin role and can manage other users.
2. **Invite New Users**
   - *As an Admin,* I want to invite new users by generating a one-time invitation code so that they can create accounts with specific roles.
3. **Account Reset**
   - *As an Admin,* I want to reset user accounts by generating a one-time password with an expiration time so that users can regain access securely.
4. **Delete User Account**
   - *As an Admin,* I want to delete user accounts after a confirmation step so that unwanted accounts can be removed from the system.
5. **View and Manage User Accounts**
   - *As an Admin,* I want to view a list of all user accounts with their roles and personal information so that I can manage users effectively.
   - *As an Admin,* I want to add or remove roles for users so that they can perform the appropriate actions within the system.
6. **Log Out**
   - *As an Admin,* I want to be able to log out of the system so that I can securely end my session.

General User Stories

1. **Account Creation via Invitation**
    - *As a user invited to the system via a one-time code,* I want to create my account by specifying a username and password so that I can start using the system with the roles assigned to me.
2. **Password Confirmation**
    - *As a user creating a new account,* I want to enter my password twice so that the system can confirm that my passwords match and ensure security.
3. **Finish Setting Up Account**
    - *As a user logging in for the first time,* I want to provide my email and full name (first, middle, last, and optionally a preferred first name) so that my account is complete and personalized.
4. **User Home Page and Log Out**
    - *As a user,* I want to be directed to my home page based on my role so that I can start using the features specific to my role, with an option to log out.
    - *As a user,* I want to be able to log out of the system so that I can securely end my session.

Multi-role User Stories

1. **Role Selection on Login**
    - *As a user with multiple roles (e.g., Admin, Student, Instructor),* I want to choose the appropriate role when logging in so that I can access the relevant features for my current role.
2. **Single Role Selection**
    - *As a user with only one role,* I want to be automatically taken to the home page for my role so that I can start using the system without selecting a role.

Phase 2 user stories:

Admin/Instructor user stories

1. **Article Identifier**
    a. As an instructor/administrator creating articles I want each article to have a long number based identifier so duplicate articles can be detected
2. **Backup and restore articles**
    a. As an instructor/administrator creating articles, I want to be able to backup and restore articles or groups of articles. When I execute the restore command. I should have the option to remove all existing help articles or merge the backed up articles with current articles.

3. **Groups of help articles**
    a. As an instructor/administrator creating articles, I want to be able to put articles into groups. I want to be able to backup groups of articles so only articles in the article are backed up. An article can belong to multiple groups.
4. **List of articles**
    a. As an instructor/administrator creating articles, I want to be able to create, update, view, and delete articles.
5. **Create, delete, view and update**
    a. As an instructor/administrator creating articles, I want to be able to list all articles, subsets of articles in a group or multiple groups.

Phase 3 user stories:

Admin user stories:

1. **Backup and restore Articles differently**
    a. An admin can backup and restore all articles, one or more groups of articles, or one or more special groups of articles
2. **Group article access**
    a. An admin may be given access over general article groups and special access article groups
3. **Add, view, and remove users from groups**
    a. An admin should be able to add, view, and remove any type of user from general groups.
4. **Creating and deleting help articles**
    a. Admins can create and delete help articles

Instructor user stories:

1. **Instructor search and viewing of articles**
    a. An instructor has the same searching and viewing actions as student user as listed in student user stories
2. **Instructor actions involving changing articles**
    a. An instructor can create, delete, view, and edit articles, general groups of articles and special access groups of articles.
3. **Group of articles restore and back up**
    a. Instructors can backup and restore groups of articles
4. **Instructors student access to article groups**
    a. Instructors can add, delete, and view students' help articles and general groups. Instructors with special access group rights can add, view and delete students of the groups.

Student user stories:

1. **Possible actions**

    a. The basic actions for a student are quitting the application, sending a generic message or a special message to the help system. They can also specify what they need and cannot find in the specific messages search, which is used to update a list of articles that need to be added to the system.

2. **Other possible actions**
    a. A student should be able to set the content level of articles that are returned when searching for something, and should be able to search using words, names, or phrases in the title, author, or abstract but not body. They can also search using a long identifier, and limit the search to certain groups of articles.
3. **Searching results and and actions on interacting with results**
    a. A performed search will result with active groups being specified on the first line, the number of articles that match each level, and a list of articles that match the search criteria in short form which includes a sequence number, title, author, abstract. After a search a student can do a new search, make a request to view an entire article in detail using a sequence number, or one of the other actions mentioned above and the same thing is for viewing an article.

General user stories:

1. **Special access group(SAG)**
    a. The website must have a special access groups system that has a of articles that encrypted body and limited decrypted access
2. **(SAG) List of admin access**
    a. The special access groups system has a list of admin given admin rights to create, read, update, and delete access rights of the group
3. **(SAG) First instructor**
    a. The first instructor added to a special access group is given the right to view the bodies of articles in the SAG and admin rights to the SAG
4. **(SAG) lists related to students and instructors**
    a. An SAG has a list of instructors that have been given rights to view the group's decrypted bodies of articles and a list of instructors given admin rights to the group. It also has a list of students given viewing rights to the group's to the decrypted bodies of articles
    b.

Phase 4 requirements:

1. Fix login validation with database (right now password hashes aren't matching database passwords so I have a temporary bypass)
2. Add an 'isOneTimePassword' to password characteristics (if yes it takes you to a page to reset password) (also this should have a timer so the password only lasts for 24hrs)
3. RegistrationPage and FinalRegistrationPage should be combined into one page
4. LoginPage should just have username and password prompts, aswell as the ability to input the invitationCode (only brings you to RegistrationPage if you have a valid code)
5. Add back buttons to most pages for ease of use
6. Add comments to most code

7. Fix verification code so codes actually get added to database and registering account must match codes found in database (unsure if they need to be one time use or have an expiration date)
8. Backup/Restore articles needs to be a list of existing articles (and another list of existing groups), these lists would be like a checklist where whatever ones are selected will be backed up. The restore button is a list of the articles that have been backed up and these will also be a checklist where whatever is selected will be restored and replaces the current articles information
9. Special Access Groups need to have their body information encrypted, where only select admins and instructors can view/decrypt the information (first instructor given access to a special access group is given the ability to view the information), also where the information is listed such as 'Special Access: true" add more lines, one telling you which instructors can view the decrypted information, one telling you which instructors have admin rights for that group, and list of students that can see the decrypted bodies)
10. Add search ability for instructor, and take away search ability for admin
11. Add code to automatically test stuff use homework 5
12. Make sure wireframes are made using a computer
13. Make sure there are UML diagrams
14. Add close button to login page to close the program

Architecture:
Phase 1 architecture:
<u>**Objects:**</u>

- **User**
    - Holds all identifiable information:
        - Email
            - Needs a validity checker. A regex will do fine
        - Username
        - Password
            - One time passwords and expiry dates
            - Non-string data type, Carter said we'll be going over encryption at a later date
        - Name
            - Uses the name struct for concise code
    - Role:
        - Permissions
            - Default User (Student), Instructor, Admin
            - Maybe UNIX style permission bits?
- **Topic:**
    - Useful topic identifiers
        - Topic Name
        - Topic Difficulty
            - Uses the DIFFICULTY enumeration

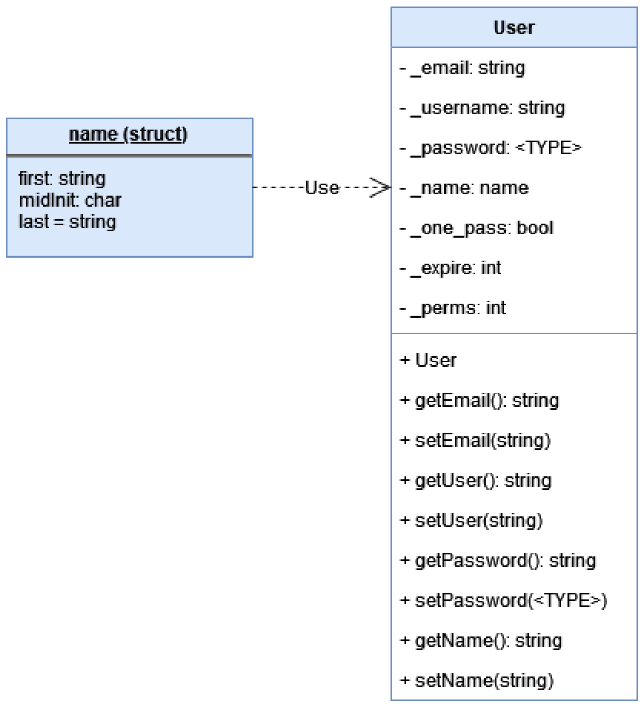
- MAYBE: topic tags

UML Class Diagrams:

Initial User UML Class Diagram:

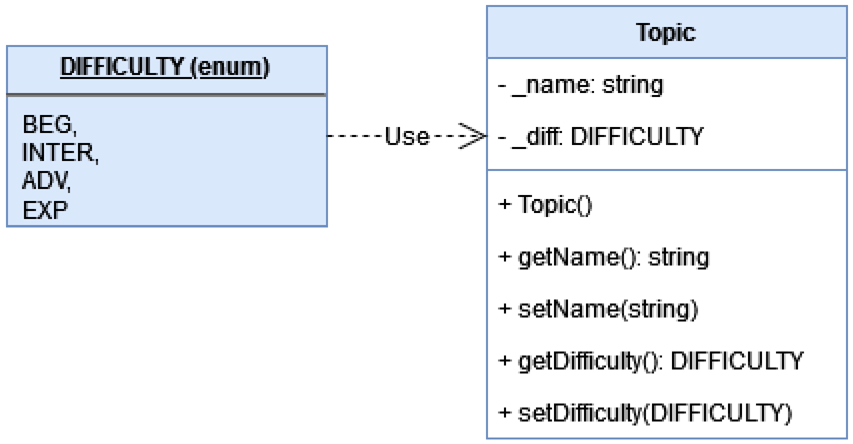
This was our initial User UML class diagram and its supporting structure.

It has storage for all needed identifiable information as well as background information for the user's permissions and password expiry.



Topic UML Class Diagram:

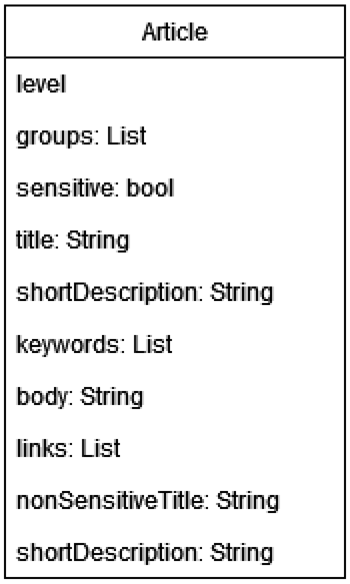
This is our initial Topic UML Diagram and supporting *DIFFICULTY* enumeration. As we reach later phases, we want to expand our Topic class to have more filters and tags for the users to accurately find their needed information.



PHASE 2 ARCHITECTURE)

We expanded on the earlier diagrams and structures and added new structures for the saving and loading of articles. Using MongoDB, we use *Documents* to store our articles

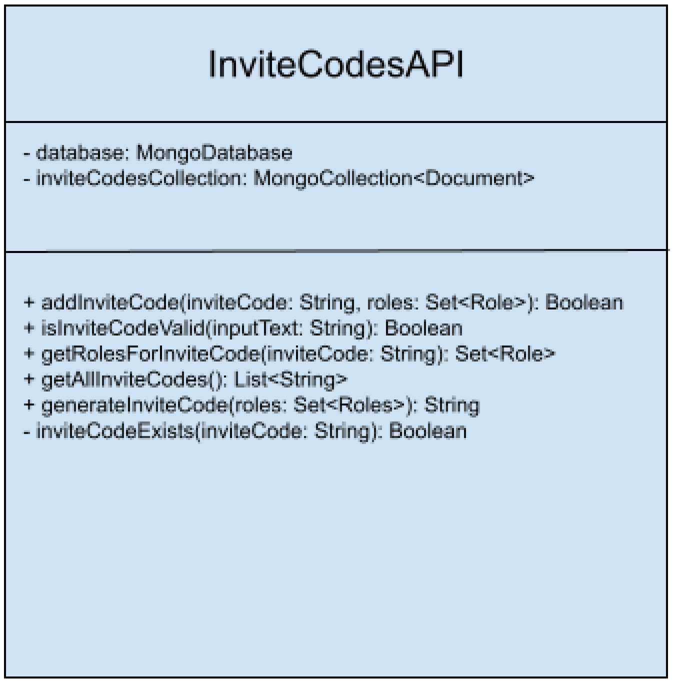
See UML below
Our articles have all relevant information stored at the same level, instead of overcomplicating sensitive articles as a subclass, just in case the sensitivity level need to be changed: so we don't have to create another *Document*.
There is a fundamental difference between document and article.
Article is the structure used at runtime, but Document is used when transitioning to and from the database
Document (of Article part diagram):



<u>PHASE 3 ARCHITECTURE)</u>
Given that this phase didn't require more objects to be made, the only changes have been to the pages and their RegisterToNavigation function, more details in the technical screencast

<u>PHASE 4 ARCHITECTURE:</u>
Phase 4 architecture hasn't changed a great deal. Most if not all of the work has been finished on the backend. We've been spending the majority of our time reworking panel structure and flow.
We added & Revamped:

- Articles API, changed how backups and restorations are made
- Special Access Groups
- Admin & Instructor panels
    - Editing
    - Deleting
    - Backup
    - Restorations


InviteCodesAPI
- database: MongoDatabase
- inviteCodesCollection: MongoCollection<Document>
+ addInviteCode(inviteCode: String, roles: Set<Role>): Boolean
+ isInviteCodeValid(inputText: String): Boolean
+ getRolesForInviteCode(inviteCode: String): Set<Role>
+ getAllInviteCodes(): List<String>
+ generateInviteCode(roles: Set<Roles>): String
- inviteCodeExists(inviteCode: String): Boolean

State Diagran
[Initial State] --> LoggedOut
LoggedOut --> LoggingIn: User enters credentials
LoggingIn --> LoggedIn: User authenticated
LoggedIn --> RoleSelection: User selects a role
RoleSelection --> RoleSelected: Role chosen
RoleSelected --> NavigatingToRolePage: System navigates to the selected role page
NavigatingToRolePage --> RolePage: User reaches the role page
RolePage --> LoggingOut: User logs out
LoggingOut --> LoggedOut: System clears user session

Design:
Data Flow:
Users data table:

users	Inputs	Outputs
Students	username	Access to help articles
Admins	password	feedback submission
Instructional Team	email	account management (admin only)

	knowledge	
	Search queries	
	feeedback	

Help System data table:

Inputs	Outputs
User requests	Help articles (filtered by knowledge level)
Ed discussion data	feedback handling
feedback	surveys

Database data table:

Stores:	Processes:
User information (account info, knowledge level)	Retrieve relevant help articles (with ability to increase/decrease information)
help articles	store feedback
feedback	update user profiles (change knowledge level)
survey results	Update information based on feedback

User Role data Table:

Student	Admin	Instructor
Search for articles	Add/edit/remove articles	Review feedback
give feedback	manage user accounts (cannot see information)	update help requests
request more/less information	Manage special access groups	Manage special access groups

Login Flow:

Users -> [Login Request] -> Help System -> [Validate] -> Database -> [Return Access] -> Users

- Admin can reset a password if needed

Search Flow:

Users -> [Search Query] -> Help System -> [Process Query] -> Database -> [Retrieve Help Articles] -> Users

- Filter results based on knowledge level

Feedback Flow:
User -> [Submit Feedback] -> Help System -> [Process Feedback] -> Database -> [Store Feedback]

Admin Article Management:
Admin -> [Add/Edit Article] -> Help System -> Database -> [Update Help Article]

Component:

User Interface:
Students: Search articles, provide feedback, and get more/less information
Admins: Manage help articles and user account information
Instructional Team: Review feedback and update help articles

Help System Core:
Handles:

- Search Functionality: fetches help articles based on queries and skill level
- Feedback processing: captures and stores feedback
- Admin management: Create, edit, or remove help articles

Authentication:

- Handles logins, password management, and roles
- Supports multi-role functionality

Database:
Stores:

- User information (login, knowledge, preferences)
- Help articles (tagged for level 1 (beginner), 2 (intermediate), 3 (advanced) or 4 (expert))
- Feedback data
- Survey results
- Handles backup and restoration functions

Ed Discussion Integration:
Fetches questions and discussions from Ed Discussion to put into help articles

Admin Tools:
Includes tools for:

- Backing up and restoring help articles and user data
- Managing users (reset password, no access to data)
- Reviewing system logs for tracking updates or issues

Database Schema:

Users:
- user_id (primary Key)
- email (unique)
- username
- password_hash (for encryption)
- is_one_time_password (boolean, if one time password is required)
- otp_expiration (time for one time validity)
- first_name
- middle_name_initial
- last_name
- preferred_name
- skill_level (1,2,3,4)

Help System:
- article_id (primary key)
- title
- content (body of help article)
- tags (comma separated list of keyword)
- author_id (foreign key to Users table, author of article)
- last_updated (minutes ago or date)
- target_skill_level (1,2,3,4)

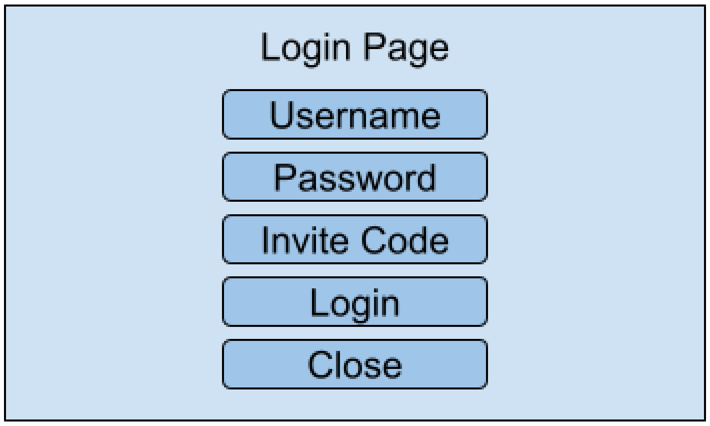
Feedback:
- feedback_id (primary key)
- user_id (foreign key to Users)
- article_id (foreign key to Help Articles)
- feedback_content
- rating
- timestamp

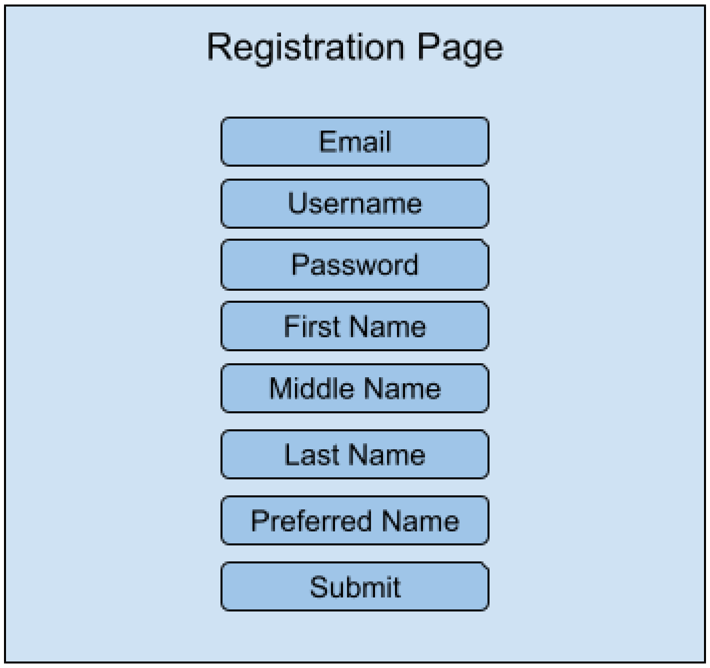
Ed Discussion Integration:
- discussion_id (primary key)
- question (text of question from Ed Discussion)
- answer (text of answer provided)

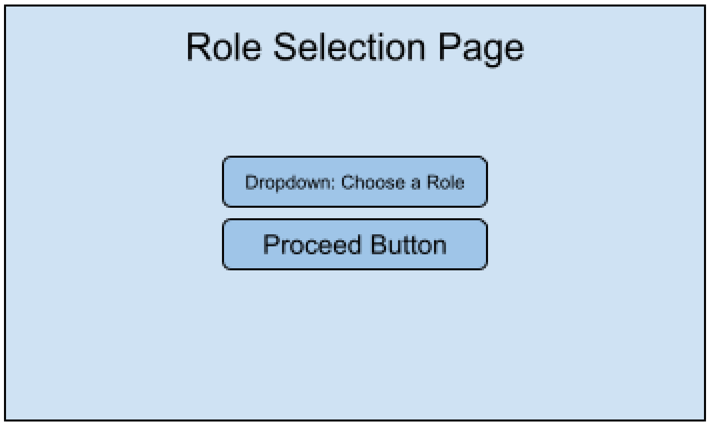
- related_articled_id (foreign key to Help Articles)

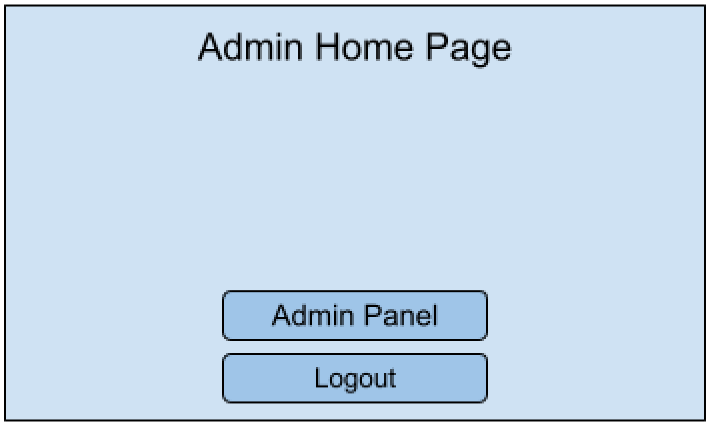
Admin Actions:
- action_id (primary key)
- admin_id (foreign key to Users, admin doing the action)
- action_type (add article, edit article, reset password)
- timestamp

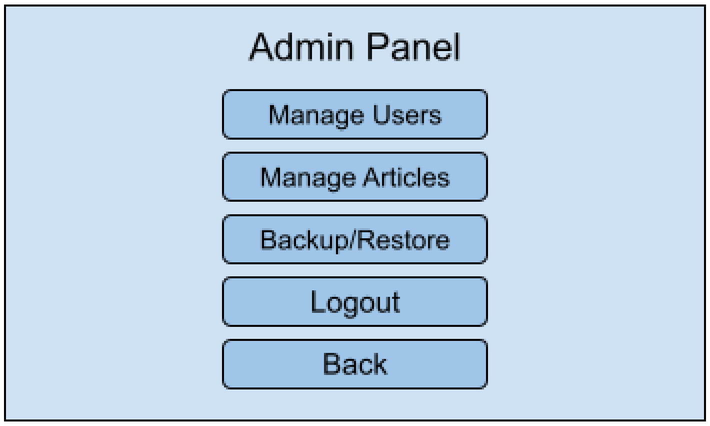
Wireframes:

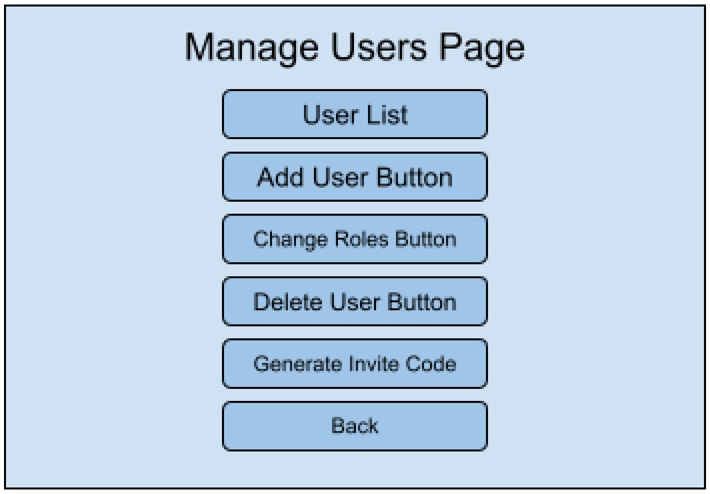


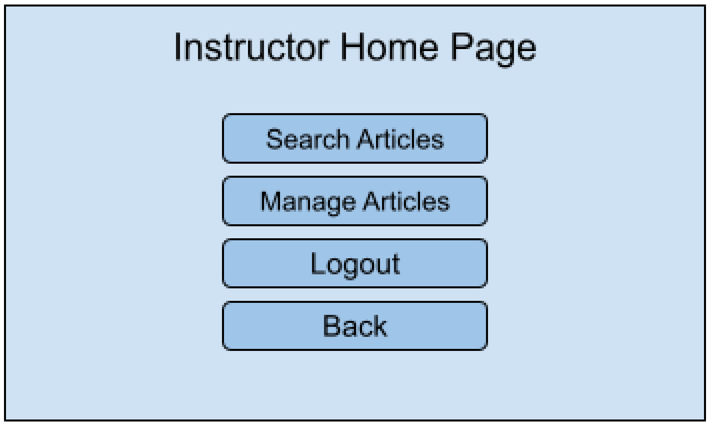


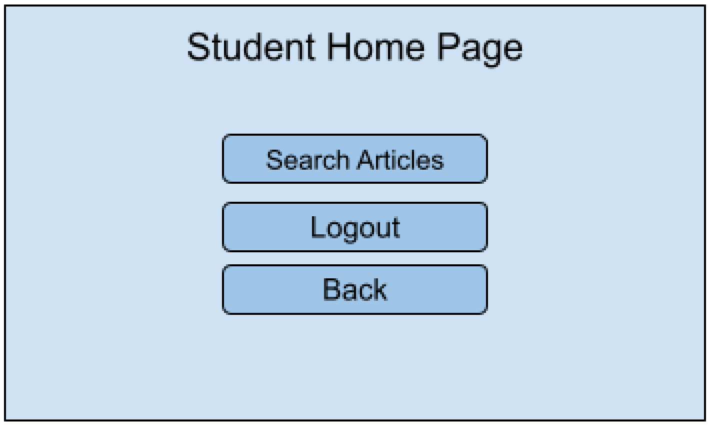

Role Selection Page
Dropdown: Choose a Role
Proceed Button


Admin Home Page
Admin Panel
Logout


Admin Panel
Manage Users
Manage Articles
Backup/Restore
Logout
Back


Manage Users Page
User List
Add User Button
Change Roles Button
Delete User Button
Generate Invite Code
Back





Class Responsibility Diagram

Class Name	Responsibilities	Collaborators
User	Store user information (username, encrypted password, email, roles), Verify user identity	UsersAPI, PasswordUtil, Role
UsersAPI	Retrieve user data from the database, add and delete users	User, Database
InviteCodesAPI	Generate and validate invite codes, store and retrieve roles for invite codes	Role, Database
Navigation	Manage scene transitions, maintain navigation history	Stage, scene
ManageUsersPage	Manage users (edit, add, delete, generate invite	UsersAPI, InviteCodesAPI, Navigation

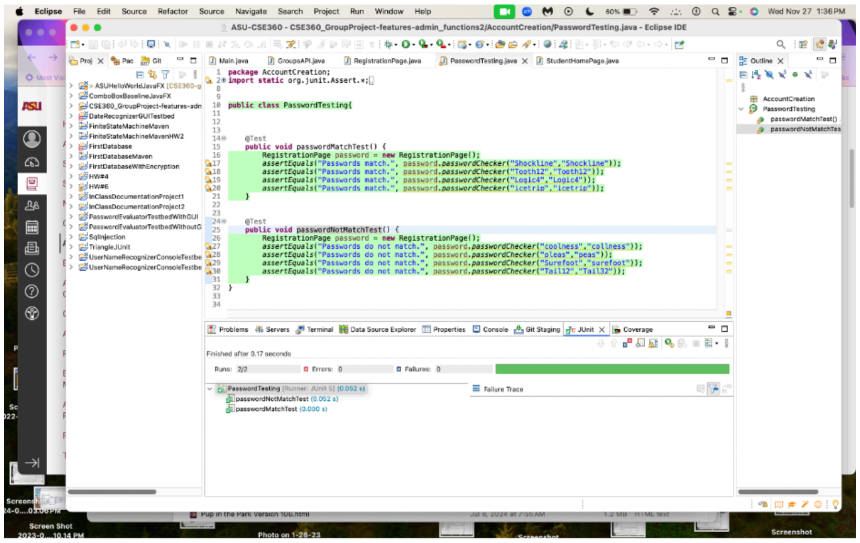
	codes)	
ManageArticlesPage	Manage articles (add, edit, delete)	ArticlesAPI, Navigation
ManageGroupsPage	Manage groups (add, edit, delete, special access)	GroupsAPI, Navigation

Class Diagram

Class	**Attributes**	**Methods**
User	**Username, password, email, role, isFinished**	**setUsername(), setPassword(), setEmail(), setRoles(), getUsername(), getRoles()**
UsersAPI		**getUserByUsername(), addUser(), deleteUserByUsername()**
InviteCodesAPI		**generateInviteCode(), isInviteCodeValid(), getRolesForInviteCode()**
Navigation	**primaryStage, scenes, currentUser**	**navigateTo(), goBack(), setCurrentUser(), getCurrentUser(), clearCurrentUser()**
ManageUsersPage	**usersList, selectedUser**	**refreshUsersList(), showAddUserDialog(), showChangeRolesDialog(), showAlert(), showConfirmationDialog()**
ManageArticlesPage	**articlesList**	**refreshArticlesList(), showAddArticleDialog(), showEditArticleDialog(), showAlert(), showConfirmationDialog()**
ManageGroupsPage	**groupsList**	**refreshGroupsList(), showAddGroupDialog(), showEditGroupDialog(), showAlert(),**

		showConfirmationDialog()
Role	**ADMIN, INSTRUCTOR, STUDENT**	

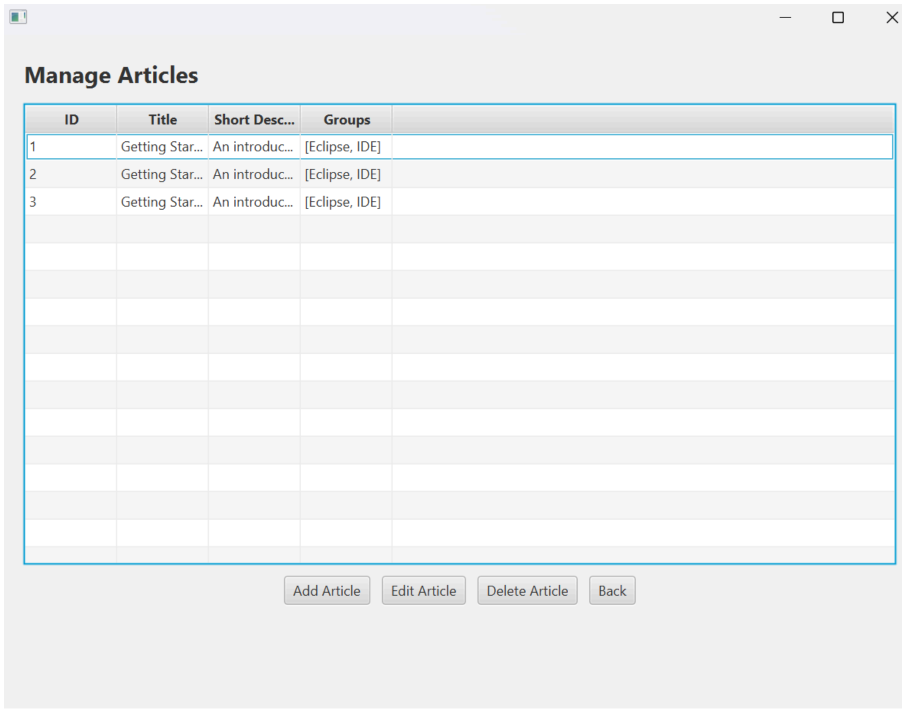
Basic and Automated Testing:
Autotest code of password tester:



```
package AccountCreation;
import static org.junit.Assert.*;


public class PasswordTesting{



    @Test
    public void passwordMatchTest() {
        RegistrationPage password = new RegistrationPage();
        assertEquals("Passwords match.", password.passwordChecker("Shockline","Shockline"));
        assertEquals("Passwords match.", password.passwordChecker("Tooth12","Tooth12"));
        assertEquals("Passwords match.", password.passwordChecker("Logic4","Logic4"));
        assertEquals("Passwords match.", password.passwordChecker("icetrip","icetrip"));
    }


    @Test
    public void passwordNotMatchTest() {
        RegistrationPage password = new RegistrationPage();
        assertEquals("Passwords do not match.", password.passwordChecker("coolness","collness"));
        assertEquals("Passwords do not match.", password.passwordChecker("pleas","peas"));
        assertEquals("Passwords do not match.", password.passwordChecker("Surefoot","surefoot"));
        assertEquals("Passwords do not match.", password.passwordChecker("Tail12","Tail32"));
    }
}
```

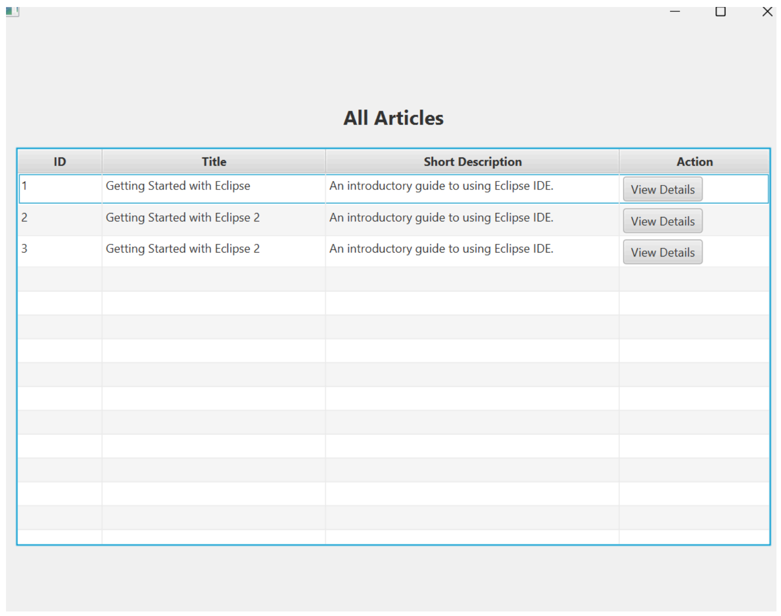
```
Finished after 0.17 seconds
Runs: 2/2    Errors: 0    Failures: 0
PasswordTesting [Runner: JUnit 5] (0.052 s)
    passwordNotMatchTest (0.052 s)
    passwordMatchTest (0.000 s)
```

Test adding, editing, and deleting articles


Manage Articles

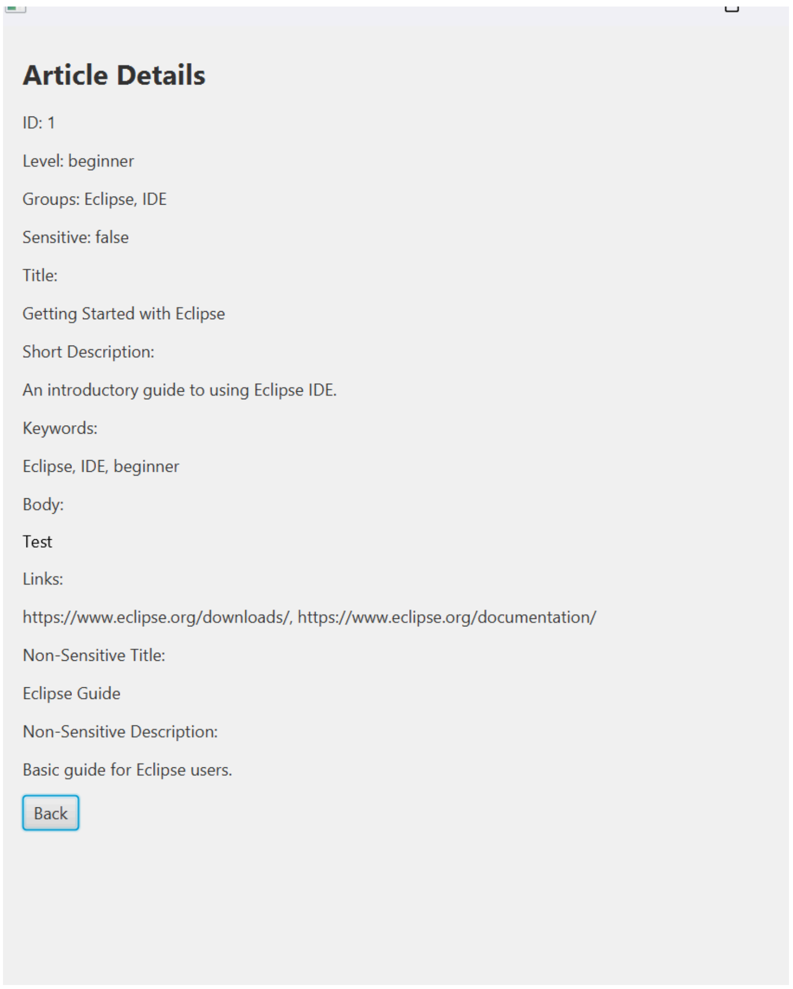
ID	Title	Short Desc...	Groups
1	Getting Star...	An introduc...	[Eclipse, IDE]
2	Getting Star...	An introduc...	[Eclipse, IDE]
3	Getting Star...	An introduc...	[Eclipse, IDE]

Tested the User list view page for all articles


All Articles

ID	Title	Short Description	Action
1	Getting Started with Eclipse	An introductory guide to using Eclipse IDE.	View Details
2	Getting Started with Eclipse 2	An introductory guide to using Eclipse IDE.	View Details
3	Getting Started with Eclipse 2	An introductory guide to using Eclipse IDE.	View Details

Tested the articles details page


Article Details

ID: 1

Level: beginner

Groups: Eclipse, IDE

Sensitive: false

Title:

Getting Started with Eclipse

Short Description:

An introductory guide to using Eclipse IDE.

Keywords:

Eclipse, IDE, beginner

Body:

Test

Links:

https://www.eclipse.org/downloads/, https://www.eclipse.org/documentation/

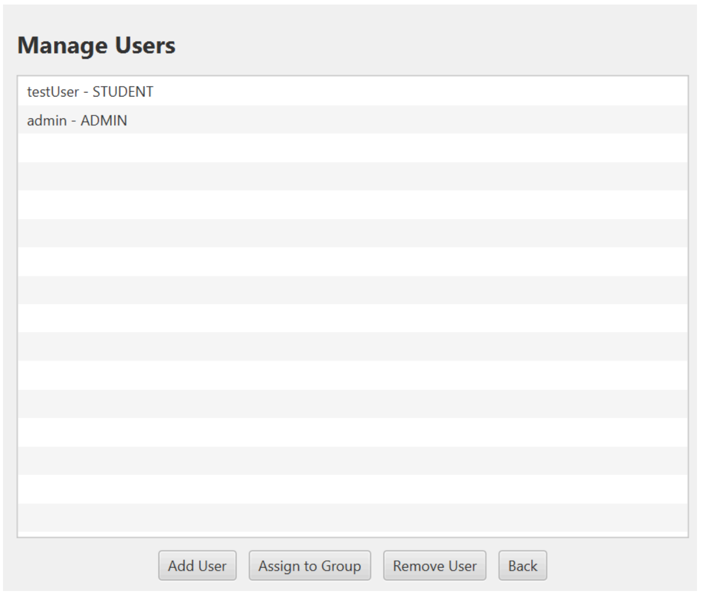
Non-Sensitive Title:

Eclipse Guide

Non-Sensitive Description:

Basic guide for Eclipse users.

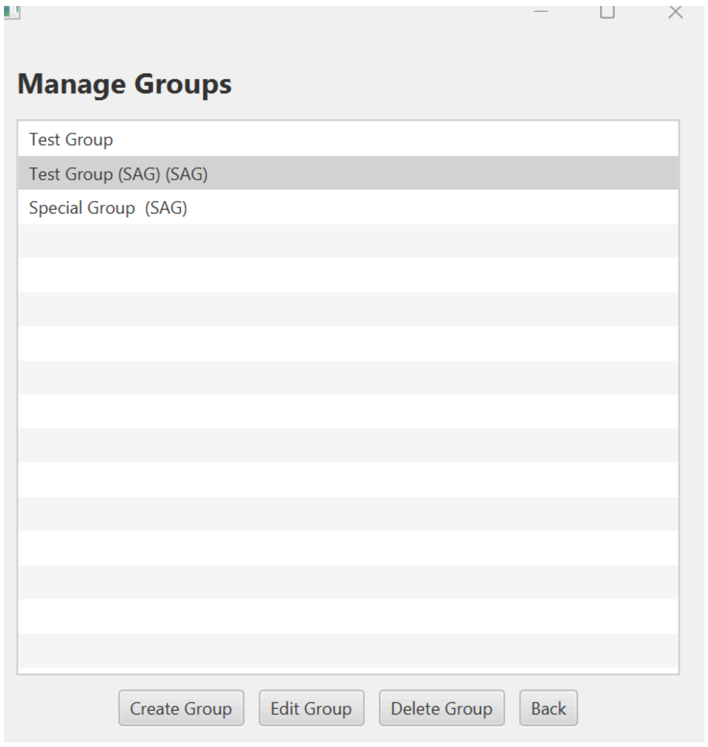
Tested the Manage Users Page by adding users, assigning to groups, and removing users


Manage Users

testUser - STUDENT

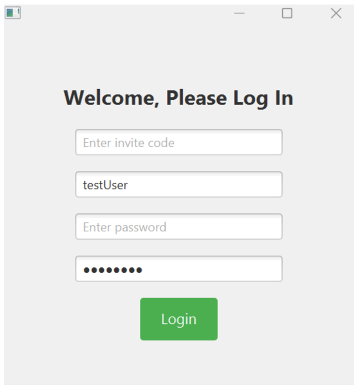
admin - ADMIN

Tested creating and removing groups including Special Access Groups (SAG)

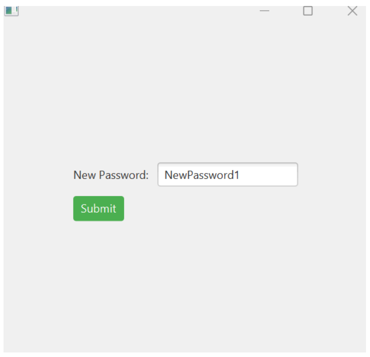

Manage Groups

Test Group
Test Group (SAG) (SAG)
Special Group (SAG)

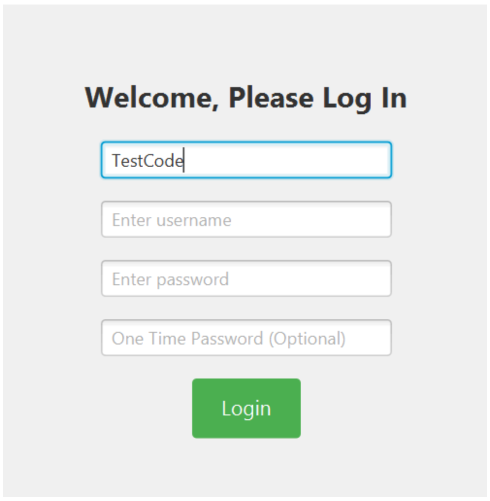
Tested logging in with One-Time Password


Welcome, Please Log In

Enter invite code
testUser
Enter password
••••••••


New Password: NewPassword1

Tested using invite code



Welcome, Please Log In

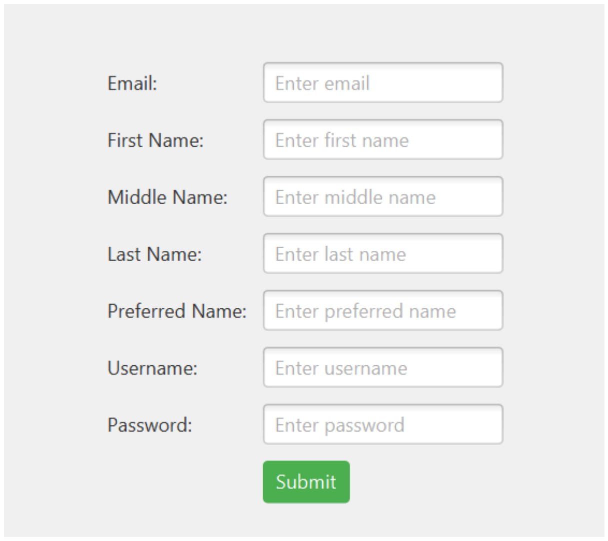
TestCode

Enter username

Enter password

One Time Password (Optional)

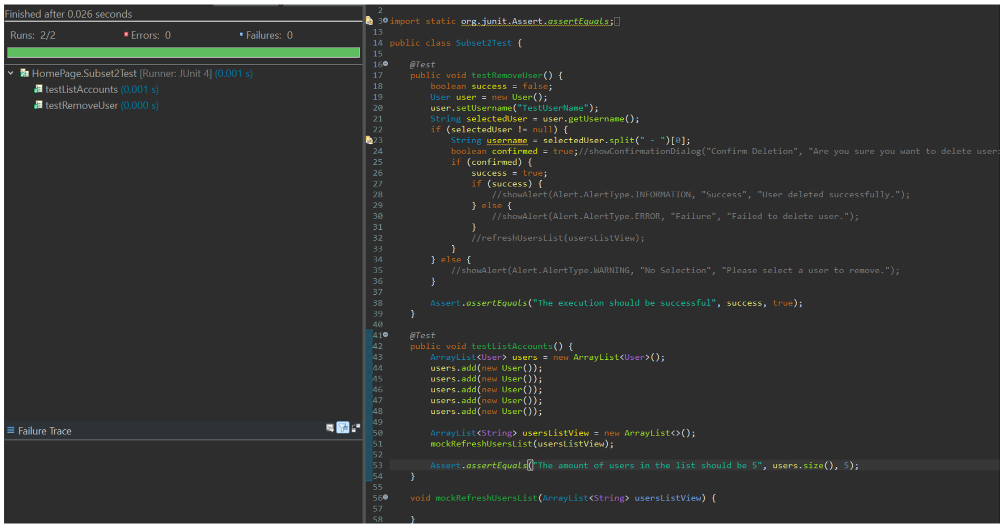
Login



Email:	Enter email
First Name:	Enter first name
Middle Name:	Enter middle name
Last Name:	Enter last name
Preferred Name:	Enter preferred name
Username:	Enter username
Password:	Enter password

Submit

Automated Unit Testing of User Features



```
Finished after 0.026 seconds
Runs: 2/2          Errors: 0          Failures: 0

HomePage.Subset2Test [Runner: JUnit 4] (0.001 s)
    testListAccounts (0.001 s)
    testRemoveUser (0.000 s)

Failure Trace
```

```java

import static org.junit.Assert.assertEquals;

public class Subset2Test {

    @Test
    public void testRemoveUser() {
        boolean success = false;
        User user = new User();
        user.setUsername("TestUserName");
        String selectedUser = user.getUsername();
        if (selectedUser != null) {
            String username = selectedUser.split(" - ")[0];
            boolean confirmed = true;//showConfirmationDialog("Confirm Deletion", "Are you sure you want to delete user
            if (confirmed) {
                success = true;
                if (success) {
                    //showAlert(Alert.AlertType.INFORMATION, "Success", "User deleted successfully.");
                } else {
                    //showAlert(Alert.AlertType.ERROR, "Failure", "Failed to delete user.");
                }
                //refreshUsersList(usersListView);
            }
        } else {
            //showAlert(Alert.AlertType.WARNING, "No Selection", "Please select a user to remove.");
        }

        Assert.assertEquals("The execution should be successful", success, true);
    }

    @Test
    public void testListAccounts() {
        ArrayList<User> users = new ArrayList<User>();
        users.add(new User());
        users.add(new User());
        users.add(new User());
        users.add(new User());
        users.add(new User());

        ArrayList<String> usersListView = new ArrayList<>();
        mockRefreshUsersList(usersListView);

        Assert.assertEquals("The amount of users in the list should be 5", users.size(), 5);
    }

    void mockRefreshUsersList(ArrayList<String> usersListView) {

    }
```

Both screencasts are not up to date with the code.

Screencast 1, technical screencast: Ethan, Matthew, and Kathan
Different parts, scheduling issues etc…

Part 1: https://drive.google.com/file/d/1z-9RgEYM7M8eNHN8S9Xw3g9zyErbyptZ/view?usp=sharing
Part 2: https://drive.google.com/file/d/1IngkdBTG6lfS0_eyg5oTFRtDTzDvMjO2/view?usp=sharing
Part 3:
https://asu.zoom.us/rec/share/t-afnCM3xzhPTkHdnKK5Wz3BhG6GYIYhfm3lYABo5uO8wgTVWsq9AdDYlr4vPEyW.hjjEJwniqp2hnwYM?startTime=1733538261000

Passcode: V7=^eZ1U
Basic plan Outline for Screencast 1:
Ethan, Matthew, and Kathan were assigned screencast1, the technical screencast. Ethan covered part1 of the screencast and covered some of the material. Matthew covered part2 of the screencast and covered some more of the material. Kathan covered part3 of the screencast and covered the rest of the material and some repeat material.

Screencast 2, how to use screencast: with Christian, Ryan, and Botirbek.
Part1:https://drive.google.com/file/d/1uUApgGqzAVoE0cE9T5FSNZOo_1t6ZHzW/view?usp=sharing
Part2: https://drive.google.com/file/d/1J6aiHaZ0iJttPuaaGa_mkLibRc80gR-t/view?usp=sharing

Basic plan Outline for Screencast 2:
Me, Ryan, And Botirbek were assigned screencast2, the how to use screencast. The plan for me and Ryan to do a part of screencast2 where we each took turns talking about certain parts of the code executions we were running. Then we had Botirbek do the second part of the screencast where we had him talk about code parts we suggested.

GitHub URL:
https://github.com/Matthew-Ager1/CSE360_GroupProject.git

Credit Page

Team Member Name	Contributions
Christian Halas	I created a new PDF document and created it based on our layout of the previous documents. I Looked at the results of the previous phases and made sure the guys knew what we needed to change. I moved the search feature from the admin to the instructor home page and added a close button to the login page. I last did my part on the second screencast.
Ethan Rudy	Kathan and I worked together on the updated backup functionality Architecture Section Screencast #1
Ryan Reny	I have done most of the Design segments of our project such as wireframes and all of the diagrams. I have handled the search result page as well as the ability to include feedback. I worked with Matt to implement the admin panel functions (resetting users, deleting users, generating codes, etc.), I worked on the pages/logic to create/verify a user and assign their roles. I dealt with the invite codes.
Matthew Ager	Creates InviteCodes API and updated UsersAPI to handle new changes including adding Invite codes and One-Time Passwords. Combined the registration pages into one page to handle making new accounts. Updated the login page to handle invite codes and one time passwords.
Kathan Parag Shah	Worked with Ethan on backup and restore article code. Screencast 1 Part 3
Botirbek Omonov	I was working on creating/deleting/updating/listing function that was missed in Phase 2. While I was working, one of my teammates, Matthew, made a huge change which include the functions I was working on. How-to Screencast