Adaptive Learning Support Platform (ALSP) - CSE 360 Help System

Gayathri Kota
Saisrivathsan Manikandan
Shaashvat Mittal
Naga Sathvik Kommareddy
Mukesh Tulluru

[11/19/2024]

Project Overview

The CSE 360 Help System is a role-based assistance application designed to support students, instructors, and admins in the CSE 360 course at ASU. It provides customized assistance to ensure that students receive guidance suitable for their skill level and specific needs, enhancing accessibility in a diverse learning environment.

Building on the foundational functionalities of Phase One which established secure identity management and role-based access control Phase Two focuses on data management and personalized support. In this phase, admins and instructors gain capabilities to create, update, and organize help content, making it easier to manage and deliver relevant assistance as required.

Phase One of the project focused on:

First Admin Setup:

- If no other users are currently on the system, the first person to log in automatically assumes the Admin role. They'll need to set up their account with a username and password for security. After that, they're taken straight to the login page.

Secure Account Creation:

- Everyone - Admins, Students, and Instructors - has to create an account with a username and password to keep things secure and reliable. To prevent any mistakes, password confirmation is required during the setup process.

Finish Account Setup:

- After the first login, users are taken to a "Finish setting up your account" page, where they need to fill in some extra details like their email, first name, middle name, last name, and an optional preferred first name. This helps the system gather everything it needs to personalize interactions for each user.

Role-Based Access Control:

- The system has three main roles: Admin, Student, and Instructor. If a user has more than one role, they'll be asked to choose which role to use when logging in. If they have just one role, they're automatically taken to the home page for that role. In Phase One, the home pages for Students and Instructors only have a logout option, but Admins get extra features for managing users.

Admin Capabilities:

- **Invite New Users**: Admins can invite new users by creating a one-time code. This lets the invited users set up their account with a username and password. Once they've done that, they're taken back to the login page to sign in.
- **Reset User Account**: Admins can reset a user's account by generating a one-time password that expires after a certain time. The user has to use this one-time password to log in and set a new password before they can access the system again.
- **Delete User Account**: Admins can permanently delete a user's account, but to prevent any mistakes, the system asks for an "Are you sure?" confirmation before the account gets deleted.
- **View and Modify User Roles**: Admins can see a list of users with their usernames, names, and roles. They can also add or remove roles for any user, making it easy to manage roles as needed.

Logging In and Logging Out:

- Users can log in with their credentials and are taken to the right home page based on their role. Each role has its own access rights, and everyone can log out from their home page.

Conclusion: In **Phase One**,We've successfully set up the core of the CSE 360 Help System with secure identity management and role-based access control. These features will serve as the base for future development, where we'll focus on adding content management and personalized support tailored to each student's experience and needs.

Phase Two of the project focused on:

Phase Two builds on these foundational features, adding functionalities for data management, content organization, and advanced access controls to improve resource handling and support delivery. Key features added include:

- **Unique Identifier Assignment:** Each help article now receives a unique long integer ID, preventing duplicate entries and ensuring accurate tracking.
- **Article Management:** Admins and instructors can create, update, and delete articles as needed, allowing for flexible content management.

- **Backup and Restore Options:** Articles can be backed up to an external file, with options to merge or replace existing content during restore based on unique identifiers, ensuring secure data handling.
- **Article Grouping:** Articles can be organized into topic-based groups, supporting easy content management by category.
- **Search and Display**: Articles are now searchable by keywords, allowing users to quickly locate relevant resources, while descriptions and titles provide filtering options for targeted searches.
- **Content Privacy and Accessibility:** Sensitive information is limited to specific roles, ensuring that content access aligns with role-based permissions.

Conclusion: In Phase Two, we improved the CSE 360 Help System's flexibility and utility by adding unique identifiers, backup and restore features, and article grouping functionalities. These enhancements make the help system more organized, secure, and practical for everyone involved in the CSE 360 course.

Phase Three :

Phase Three enhances the platform by addressing security, usability, and administrative control requirements:

- Encryption of Sensitive Data:
    - The body of sensitive articles is encrypted, ensuring secure storage and retrieval.
    - Even if system memory is analyzed, encrypted content remains secure.
- Special Access Groups:
    - Admins can create, manage, and delete special access groups.
    - Special access groups contain sensitive articles with encrypted content.
    - Admins do not automatically have the right to view these articles unless explicitly granted access.
    - The first instructor added to a group gains admin rights and the ability to view article content within that group.
    - Students, instructors, and admins can have role-specific access to group articles.
- Enhanced Student Interface:
    - Students can:
        - Search articles using keywords in titles, authors, or abstracts.
        - Filter results by content levels (beginner, intermediate, etc.) or specific groups.
        - Send feedback for missing or unclear articles.
    - Search results display the group, content levels, and a short summary of matching articles.
    - Students can view article details or refine their searches.

- Additional Instructor Capabilities:
    - Instructors can:
        - Manage special access groups.
        - Assign or revoke access rights for students and other instructors.
        - Backup and restore articles or groups, maintaining encrypted content.
- JUnit Automated Testing:
    - Non-UI components have automated test coverage.
    - Ensures functionality and reliability across all features.
- Backup and Restore Enhancements:
    - Backups include encrypted article bodies to preserve security.
    - Restores maintain encryption and prevent data duplication.

Updates in Phase Four :

Phase Four focused on ensuring that all requirements and previous feedback from graders were implemented. After a thorough check and testing of requirements, since we received no feedback from graders and got a full score for all the previous phases no change was required to our project.

Conclusion:

We have developed a comprehensive help system tailored to meet the unique needs of ASU CSE 360 students, offering a reliable, efficient, and personalized approach to accessing software engineering information. This system has been designed to address the diverse backgrounds and experiences of students in the course, providing resources that are neither too simplistic nor overly complex, but instead dynamically suited to the user's expertise and current challenges.

<u>Requirements (User Stories)</u>

<u>Phase One:</u>

Account Creation:

- As a user, I want to be invited to the system using a one-time code so that I can create my account securely.
- As a user, I want to enter my invitation code and create my account by providing my username and password so that I can access the system.

- As an admin, I want to assign roles (e.g., Student, Admin, Instructor) to users when inviting them so that their access is limited to appropriate features.

Login and Role Selection:

- As a user, I want to log in using my username and password so that I can access my account.
- As a user with multiple roles, I want to select the role I wish to use for this session so that I can access the appropriate features.
- As a user with only one role, I want to be automatically redirected to the home page for my role so that I can start using the system quickly.

Password Management:

- As a user, I want to reset my password if I forget it by requesting a one-time password via email so that I can regain access to my account.
- As a user, I want to enter a new password after receiving the OTP so that my account remains secure.
- As a user, I want to be required to confirm my new password by entering it twice so that I don't mistakenly set an incorrect password.

Account Setup (Post-Login):

- As a user, I want to provide my email address and full name after logging in for the first time so that my account setup is complete.
- As a user, I want to enter my full name (first, middle, last) and optionally a preferred first name so that the system can address me correctly.

Phase Two:

Help Article Management:

- As an admin/instructor, I want to create, update, and delete help articles with detailed fields (title, description, keywords, body, and references) to keep resources relevant and organized for students.
- As an admin/instructor, I want each help article to have a unique identifier to prevent duplicate entries and simplify management.

Backup and Restore:

- As an admin/instructor, I want to back up articles to an external file and have options to merge or replace current articles during a restore, allowing me to manage data securely.

- As an admin/instructor, I want articles with the same unique identifier to not be duplicated during a restore, ensuring data consistency.

Create and Delete Groups:

- As an admin/instructor, I can create groups (e.g., Eclipse, GitHub) to organize articles by category.
- As an admin/instructor, I can delete groups to manage and streamline article organization.

Article Grouping and Organization:

- As an admin/instructor, I want to categorize articles by groups for easier access and organization.
- As an admin/instructor, I want to list help articles filtered by one or multiple groups to quickly locate specific resources.

Search and Display:

- As a student, I want to search for help articles using keywords to quickly find the resources I need.
- As a student, I want to view a brief description of each article in search results so that I can decide which resource is most relevant.

Role-Based Content Access:

- As an admin/instructor, I want to limit access to specific articles based on user roles, ensuring sensitive information is protected.
- As a student, I want to view only articles relevant to my role, giving me easy access to appropriate resources.

Phase Three :

Encryption and Privacy

- As a developer, I want to encrypt the body of sensitive articles to ensure private information remains secure even if system memory is accessed or dumped.
- As a user, I want access to encrypted articles controlled by role-based permissions, so that only authorized users can view them.

Special Access Groups

- As an admin, I want to create, manage, and delete special access groups to secure and organize proprietary or sensitive articles.

- As an admin, I want the ability to assign admins and instructors as group administrators without automatically granting them access to article content.
- As an instructor, I want to gain viewing and administrative rights to a special access group only if explicitly granted or if I am the first instructor added to the group.
- As a student, I want to view decrypted article bodies in special access groups when explicitly granted access by an admin or instructor.

Student Interface Enhancements

- As a student, I want to search for help articles using keywords in titles, abstracts, and authors to quickly find resources relevant to my needs.
- As a student, I want to filter search results by content level (e.g., beginner, intermediate, advanced, expert) to find articles suited to my current knowledge.
- As a student, I want the ability to specify a group filter (e.g., help articles for a specific assignment) to narrow my search scope.
- As a student, I want to send feedback messages (generic or specific) to report missing or unclear articles so that the instructional team can improve the available resources.
- As a student, I want the system to track my search queries when I report missing articles so that new articles can be developed based on common needs.

Instructor Role Enhancements

- As an instructor, I want the ability to search for articles with the same features as students but with additional rights to access special access groups.
- As an instructor, I want to view the details of help articles from search results to review their content.
- As an instructor, I want to manage (create, update, delete) special access groups to ensure sensitive articles are organized and accessible to authorized users.
- As an instructor, I want to back up and restore articles within special access groups while ensuring encryption is preserved.

Admin Role Enhancements

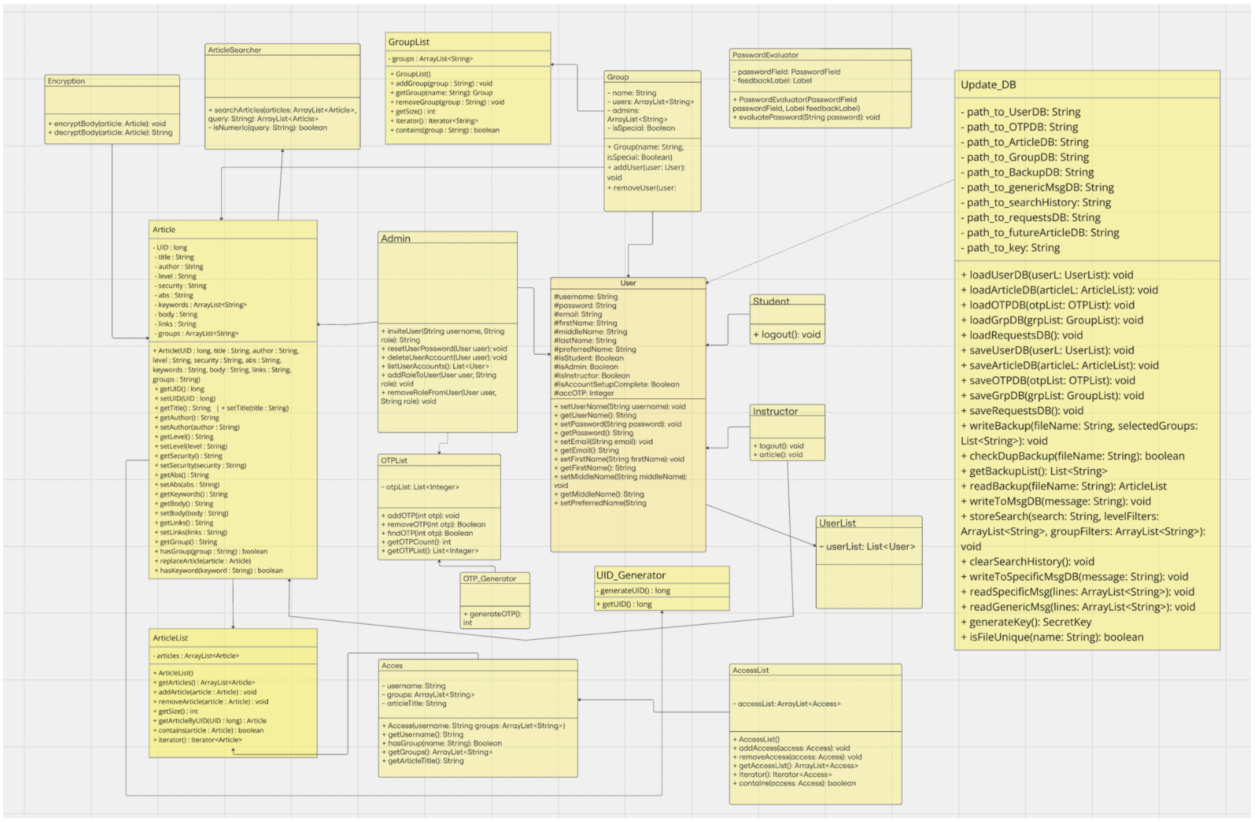
- As an admin, I want to create, delete, and manage special access groups to organize sensitive articles effectively.
- As an admin, I want to back up and restore all groups and articles, including encrypted articles, while maintaining role-based access controls.

Updates in Phase Four :

Phase Four focused on ensuring that all requirements and previous feedback from graders were implemented. After a thorough check and testing of requirements, since we received no feedback from graders and got a full score for all the previous phases no change was required to our project.

Architecture

Link to UML:
https://miro.com/app/board/uXjVLXYd1CM=/?share_link_id=228700570795



Encryption
+ encryptBody(article: Article): void
+ decryptBody(article: Article): String
ArticleSearcher
+ searchArticles(articles: ArrayList<Article>, query: String): ArrayList<Article>
- isNumeric(query: String): boolean
GroupList
- groups : ArrayList<String>
+ GroupList()
+ addGroup(group : String) : void
+ getGroup(name: String): Group
+ removeGroup(group : String) : void
+ getSize() : int
+ iterator() : Iterator<String>
+ contains(group : String) : boolean
Group
- name: String
- users: ArrayList<String>
- admins: ArrayList<String>
- isSpecial: Boolean
+ Group(name: String, isSpecial: Boolean)
+ addUser(user: User): void
+ removeUser(user:
PasswordEvaluator
- passwordField: PasswordField
- feedbackLabel: Label
+ PasswordEvaluator(PasswordField passwordField, Label feedbackLabel)
+ evaluatePassword(String password): void
Update_DB
- path_to_UserDB: String
- path_to_OTPDB: String
- path_to_ArticleDB: String
- path_to_GroupDB: String
- path_to_BackupDB: String
- path_to_genericMsgDB: String
- path_to_searchHistory: String
- path_to_requestsDB: String
- path_to_futureArticleDB: String
- path_to_key: String
+ loadUserDB(userL: UserList): void
+ loadArticleDB(articleL: ArticleList): void
+ loadOTPDB(otpList: OTPList): void
+ loadGrpDB(grpList: GroupList): void
+ loadRequestsDB(): void
+ saveUserDB(userL: UserList): void
+ saveArticleDB(articleL: ArticleList): void
+ saveOTPDB(otpList: OTPList): void
+ saveGrpDB(grpList: GroupList): void
+ saveRequestsDB(): void
+ writeBackup(fileName: String, selectedGroups: List<String>): void
+ checkDupBackup(fileName: String): boolean
+ getBackupList(): List<String>
+ readBackup(fileName: String): ArticleList
+ writeToMsgDB(message: String): void
+ storeSearch(search: String, levelFilters: ArrayList<String>, groupFilters: ArrayList<String>): void
+ clearSearchHistory(): void
+ writeToSpecificMsgDB(message: String): void
+ readSpecificMsg(lines: ArrayList<String>): void
+ readGenericMsg(lines: ArrayList<String>): void
+ generateKey(): SecretKey
+ isFileUnique(name: String): boolean
Article
- UID : long
- title : String
- author : String
- level : String
- security : String
- abs : String
- keywords : ArrayList<String>
- body : String
- links : String
- groups : ArrayList<String>
+ Article(UID : long, title : String, author : String, level : String, security : String, abs : String, keywords : String, body : String, links : String, groups : String)
+ getUID() : long
+ setUID(UID : long)
+ getTitle() : String | + setTitle(title : String)
+ getAuthor() : String
+ setAuthor(author : String)
+ getLevel() : String
+ setLevel(level : String)
+ getSecurity() : String
+ setSecurity(security : String)
+ getAbs() : String
+ setAbs(abs : String)
+ getKeywords() : String
+ getBody() : String
+ setBody(body : String)
+ getLinks() : String
+ setLinks(links : String)
+ getGroup() : String
+ hasGroup(group : String) : boolean
+ replaceArticle(article : Article)
+ hasKeyword(keyword : String) : boolean
Admin
+ inviteUser(String username, String role): String
+ resetUserPassword(User user): void
+ deleteUserAccount(User user): void
+ listUserAccounts(): List<User>
+ addRoleToUser(User user, String role): void
+ removeRoleFromUser(User user, String role): void
User
#username: String
#password: String
#email: String
#firstName: String
#middleName: String
#lastName: String
#preferredName: String
#isStudent: Boolean
#isAdmin: Boolean
#isInstructor: Boolean
#isAccountSetupComplete: Boolean
#accOTP: Integer
+ setUserName(String username): void
+ getUserName(): String
+ setPassword(String password): void
+ getPassword(): String
+ setEmail(String email): void
+ getEmail(): String
+ setFirstName(String firstName): void
+ getFirstName(): String
+ setMiddleName(String middleName): void
+ getMiddleName(): String
+ setPreferredName(String
Student
+ logout(): void
Instructor
+ logout(): void
+ article(): void
OTPList
- otpList: List<Integer>
+ addOTP(int otp): void
+ removeOTP(int otp): Boolean
+ findOTP(int otp): Boolean
+ getOTPCount(): int
+ getOTPList(): List<Integer>
OTP_Generator
+ generateOTP(): int
UID_Generator
- generateUID() : long
+ getUID() : long
UserList
- userList: List<User>
ArticleList
- articles : ArrayList<Article>
+ ArticleList()
+ getArticles() : ArrayList<Article>
+ addArticle(article : Article) : void
+ removeArticle(article : Article) : void
+ getSize() : int
+ getArticleByUID(UID : long) : Article
+ contains(article : Article) : boolean
+ iterator() : Iterator<Article>
Acces
- username: String
- groups: ArrayList<String>
- articleTitle: String
+ Access(username: String groups: ArrayList<String>)
+ getUsername(): String
+ hasGroup(name: String): Boolean
+ getGroups(): ArrayList<String>
+ getArticleTitle(): String
AccessList
- accessList: ArrayList<Access>
+ AccessList()
+ addAccess(access: Access): void
+ removeAccess(access: Access): void
+ getAccessList(): ArrayList<Access>
+ iterator(): Iterator<Access>
+ contains(access: Access): boolean

The Phase Three UML diagram shows how the help system for the CSE 360 course has been enhanced with a focus on security, access control, and improved user interfaces.

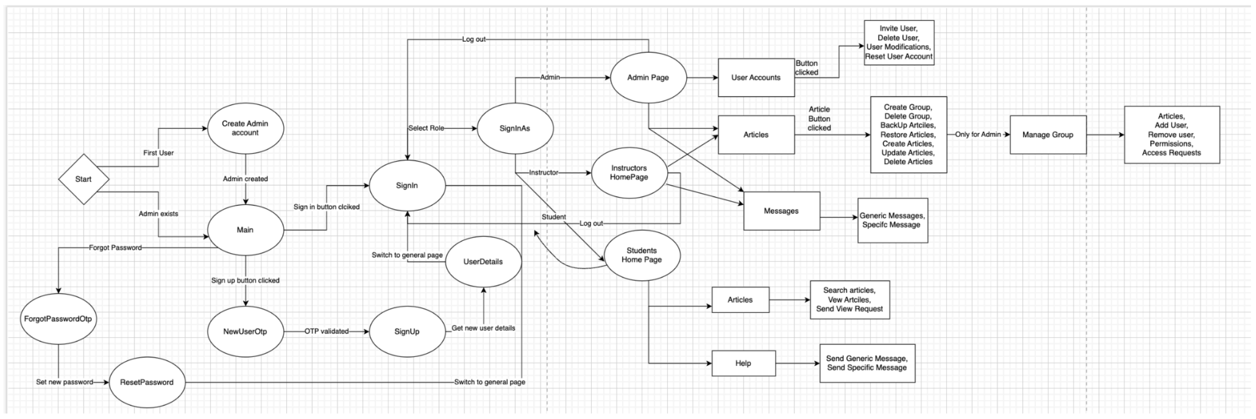
At the center of the system is still the User class, which holds basic information about each user (like username, password, email, etc.) and flags to indicate their role (Student, Instructor, or Admin). Each role has its specific permissions:

- Admins have expanded powers, including managing users and articles, creating groups, resetting passwords, and managing access protocols.
- Instructors can securely create, organize, and encrypt articles for their groups.
- Students can search for and view articles based on their permissions and request assistance if needed.

The new Encryption class ensures that sensitive data, such as article content, is securely encrypted before storage and decrypted only for authorized users. The AccessList class manages permissions for articles and groups, ensuring that only users with the appropriate access can view certain content.

The Article class continues to store detailed information for each article, with the body now encrypted for security. The ArticleList class enables secure retrieval and organization of articles. The GroupList class allows grouping articles by categories or topics, while the new OTPList and OTP_Generator classes enhance security by managing and generating one-time passwords for user verification.

These new features improve privacy and ensure secure data management while maintaining the system's usability for all roles.



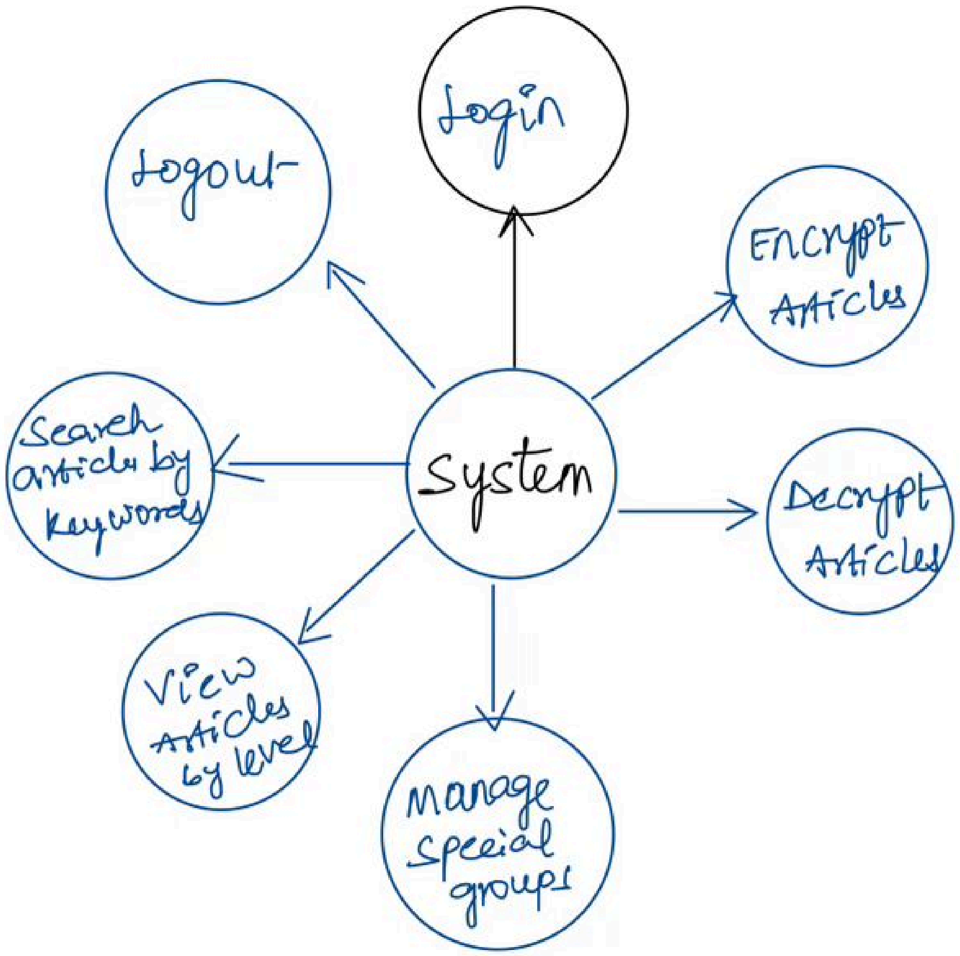
Application Flow diagram that describes how a user might navigate the application. The arrows define the action or the scenario and the Circle icon denotes the java class file which displays the current GUI of the window.

Link:

https://drive.google.com/file/d/1gLhN4WF_CvqZZOGD3xIURJZKMKYa9Ibt/view?usp=sharing

Design

Use Case diagram



Use Case 1: Admin Manages Special Access Groups

- **Pre-condition:** The Admin is logged into the system.
- **Primary Pathway (Happy Path):**
  1. The Admin selects "Manage Special Access Groups" from the dashboard.
  2. The system displays existing groups and an option to create a new group.
  3. The Admin enters group details, including name and type (e.g., sensitive/proprietary).
  4. The system creates the group and confirms the action.
- **Post-condition:** A new special access group is created, or an existing group is updated or deleted.

- **Alternate Pathway:** If the group name already exists, the system prompts the Admin to choose a unique name.
- **Exception Pathway:** If required fields are missing, the system prompts for completion.

Use Case 2: Admin Encrypts and Decrypts Articles

- **Pre-condition:** The Admin is logged into the system and has appropriate permissions.
- **Primary Pathway (Happy Path):**
    1. The Admin selects an article to encrypt or decrypt from the dashboard.
    2. The system prompts for confirmation and encrypts or decrypts the article content.
    3. The system updates the status of the article and confirms the action.
- **Post-condition:** The article body is encrypted or decrypted based on the Admin's request.
- **Alternate Pathway:** If the article is already in the desired state (encrypted or decrypted), the system notifies the Admin.
- **Exception Pathway:** If encryption fails due to a system error, the system logs the error and notifies the Admin.

Use Case 3: Instructor Accesses Special Access Groups

- **Pre-condition:** The Instructor is logged into the system with special access rights.
- **Primary Pathway (Happy Path):**
    1. The Instructor selects a special access group from their dashboard.
    2. The system verifies permissions and displays articles within the group.
    3. The Instructor selects an article to view its decrypted content.
- **Post-condition:** The Instructor views articles from the special access group.
- **Alternate Pathway:** If no articles are available in the group, the system notifies the Instructor.
- **Exception Pathway:** If permissions are insufficient, the system restricts access and notifies the Instructor.

Use Case 4: Student Filters Search Results by Content Level

- **Pre-condition:** The Student is logged into the system.
- **Primary Pathway (Happy Path):**
    1. The Student performs a search for help articles.
    2. The system displays an option to filter by content level (e.g., beginner, intermediate, advanced, expert).
    3. The Student selects a level or "all."
    4. The system updates the search results based on the selected filter.
- **Post-condition:** The Student views filtered search results.
- **Alternate Pathway:** If no articles match the filter, the system displays a "no results found" message.
- **Exception Pathway:** If the filter is invalid, the system prompts the Student to select a valid option.

Use Case 5: Student Sends Feedback

- **Pre-condition:** The Student is logged into the system.
- **Primary Pathway (Happy Path):**
    1. The Student selects "Send Feedback" from the interface.
    2. The system prompts for the type of feedback (generic or specific).
    3. The Student enters feedback details and submits.
    4. The system logs the feedback and associates it with the Student's recent activity.
- **Post-condition:** The feedback is logged for system review.
- **Alternate Pathway:** If feedback details are missing, the system prompts for completion.
- **Exception Pathway:** If the system fails to log the feedback, an error message is displayed.

Use Case 6: Instructor Backs Up Articles

- **Pre-condition:** The Instructor is logged into the system with appropriate permissions.
- **Primary Pathway (Happy Path):**

    1. The Instructor selects "Backup Articles" from their dashboard.
    2. The system prompts for group selection (specific group or all articles).
    3. The Instructor specifies the backup type and initiates the action.
    4. The system saves the backup and confirms success.
- **Post-condition:** The articles are backed up securely, preserving encryption.
- **Alternate Pathway:** If the destination path is invalid, the system prompts for correction.
- **Exception Pathway:** If the backup fails, the system logs the error and notifies the Instructor.

Use Case 7: Admin Restores Articles

- **Pre-condition:** The Admin is logged into the system.
- **Primary Pathway (Happy Path):**
    1. The Admin selects "Restore Articles" from the dashboard.
    2. The system prompts for a backup file.
    3. The Admin specifies whether to merge or replace current articles.
    4. The system restores the articles while retaining encryption and access permissions.
    5. The system confirms successful restoration.
- **Post-condition:** Articles from the backup are restored securely.
- **Alternate Pathway:** If a conflict arises with existing articles, the system prompts for resolution.
- **Exception Pathway:** If the backup file is invalid, the system notifies the Admin and cancels the restoration.

Class Responsibility Collaborator

1. Admin Class

Responsibility	Collaborator
Manage special access groups (create, update, delete)	SpecialAccessGroupManager
Backup and restore articles and groups	BackupHandler

Encrypt and decrypt article content	EncryptionManager
Assign admin rights for specific groups	SpecialAccessGroupManager
Create, update, and delete help articles	ArticleManager
Log out of the system	N/A

Responsibilities:

- Manage special access groups (create, update, delete).
- Backup and restore articles and groups.
- Encrypt and decrypt article content.
- Assign admin rights for specific groups.
- Create, update, and delete help articles.
- Log out of the system

Collaborators:

- **SpecialAccessGroupManager**: To manage access groups and permissions.
- **BackupHandler**: For secure backup and restoration.
- **EncryptionManager**: To encrypt and decrypt articles.
- **ArticleManager**: To handle article management.

2. Instructor Class

Responsibility	Collaborator
Create, update, and delete help articles	ArticleManager
Create and manage article groups	GroupManager
View and filter articles	SearchManager
Backup and restore articles	BackupHandler
Log out of the system	N/A

Responsibilities:

- Create, update, and delete help articles.
- Create and manage article groups.

- View and filter articles.
- Backup and restore articles.
- Log out of the system.

Collaborators:

- **ArticleManager**: To manage articles.
- **GroupManager**: To manage groups.
- **SearchManager**: To view and filter articles.
- **BackupHandler**: To back up and restore articles.

.

3. Student Class

Responsibility	Collaborator
Search for help articles	SearchManager
Filter search results by content level or group	SearchManager
Send feedback for missing or unclear articles	FeedbackManager
Access articles relevant to their role	ArticleManager
Log out of the system	N/A

Responsibilities:

- Search for help articles.
- Filter search results by level or group.
- Access articles relevant to their role.
- Send feedback for unclear or missing articles.
- Log out of the system.

Collaborators:

- **SearchManager:** To search and filter articles by level or group.
- **ArticleManager:** To view article details.
- **FeedbackManager:** To send feedback for unclear or missing content.

4. ArticleManager Class

Responsibility	Collaborator
Add, update, and delete articles	Admin, Instructor
Organize articles into groups	GroupManager
Encrypt and decrypt sensitive article content	EncryptionManager
Search and filter articles by keywords or group	SearchManager

Responsibilities:

- Add, update, and delete articles.
- Organize articles into groups.
- Encrypt and decrypt sensitive article content.
- Search and filter articles by keywords or group.

Collaborators:

- **Admin, Instructor**: To manage article creation and deletion.
- **GroupManager**: To associate articles with groups.
- **EncryptionManager**: To encrypt and decrypt articles.
- **SearchManager**: To perform advanced article searches.

5. BackupHandler Class

Responsibility	Collaborator
Backup articles and groups	Admin, Instructor
Restore articles and groups with merge/replace options	ArticleManager
Backup and restore special access groups	SpecialAccessGroupManager

Responsibilities:

- **Backup articles and groups.**
- **Restore articles and groups with merge/replace options.**
- **Backup and restore special access groups.**

Collaborators:

- **Admin, Instructor:** To initiate backup and restore actions.
- **ArticleManager:** To manage articles during backup/restore.
- **SpecialAccessGroupManager:** To back up and restore access groups.

6.GroupManager Class

Responsibility	Collaborator
Create and delete article groups	Admin, Instructor
Manage special access groups	SpecialAccessGroupManager
Organize articles by topics	ArticleManager
Assign users to groups	UserList

Responsibilities:

- Create and delete article groups.
- Manage special access groups.
- Organize articles by topics.
- Assign users to groups.

Collaborators:

- **Admin, Instructor**: To manage group creation and deletion.
- **SpecialAccessGroupManager**: To handle access group management.
- **ArticleManager**: To assign articles to groups.
- **UserList**: To assign users to groups..

7. EncryptionManager Class

Responsibility	Collaborator
Encrypt article content	ArticleManager
Decrypt article content	ArticleManager

Responsibilities:

- Encrypt article content.
- Decrypt article content.

Collaborators:

- **ArticleManager:** To handle encryption and decryption for articles.

8. SpecialAccessGroupManager Class

Responsibility	Collaborator
Create and manage special access groups	Admin
Assign users and roles within special access groups	UserList
Backup and restore special access groups	BackupHandler

Responsibilities:

- Create and manage special access groups.
- Assign users and roles within special access groups.
- Backup and restore special access groups.

Collaborators:

- **Admin**: To manage group creation and permissions.
- **UserList**: To manage group members.
- **BackupHandler**: To back up and restore access groups..

9. SearchManager Class

Responsibility	Collaborator
Perform keyword-based search for articles	Student, Instructor

Filter results by content level or group	Student, Instructor

Responsibilities:

- Perform keyword-based search for articles.
- Filter results by content level or group.

Collaborators:

- **Student, Instructor**: To search for and view articles.

10. UserList Class

Responsibility	Collaborator
Add, remove, and list users	Admin, GroupManager
Assign users to groups	GroupManager

Responsibilities:

- Add, remove, and list users.
- Assign users to groups.

Collaborators:

- **Admin**: To manage users.
- **GroupManager**: To assign users to groups.

Class Diagrams: Phase Three

1. User Class

Attributes:

- username: Unique identifier for each user.
- password: Stores the user’s password.
- email: User’s email address.

- firstName, middleName, lastName, preferredName: Personal information of the user.
- isStudent, isAdmin, isInstructor: Flags indicating the user's roles.
- isAccountSetupComplete: Boolean indicating if the account setup is completed.
- accOTP: One-Time Password for account reset.

Methods:

- setUserName(), getUserName(): Set and retrieve the username.
- setPassword(), getPassword(): Set and retrieve the password.
- setEmail(), getEmail(): Set and retrieve the email address.
- setFirstName(), getFirstName(): Set and retrieve the first name.
- setMiddleName(), getMiddleName(): Set and retrieve the middle name.
- setPreferredName(), getPreferredName(): Set and retrieve the preferred name.
- setStudent(), setAdmin(), setInstructor(): Set roles for the user.
- finishAccountSetup(), isAccountSetupComplete(): Manage account setup completion.
- setAccOTP(), getAccOTP(): Set and retrieve the OTP.

Description: Represents any system user (Admin, Instructor, or Student) with attributes for personal details, roles, and security features like OTP.

2. Student and Instructor Classes (Inherits from User)

Methods:

- logout(): Logs the user out of the system.
- searchHelpArticles(): Allows users to search articles based on keywords.
- viewArticle(): Allows users to view specific articles.
- filterSearchByLevel(): Enables filtering of search results by content level or group.
- sendFeedback(): Allows students to send feedback for missing or unclear articles.

Description: Specialized roles inheriting from User. Students can search and view articles, provide feedback, and filter content, while instructors can additionally create and manage help articles.

3. Admin Class (Inherits from User)

Methods:

- inviteUser(): Invites a new user and assigns a role.
- resetUserPassword(): Resets a user's password.
- deleteUserAccount(): Deletes a user account.
- listUserAccounts(): Lists all users in the system.
- addRoleToUser(), removeRoleFromUser(): Add or remove roles for a user.
- createHelpArticle(), deleteHelpArticle(): Create or delete articles.
- backupArticles(), restoreArticles(): Backup and restore articles and groups.
- createGroup(), deleteGroup(): Manage article groups.
- manageSpecialAccessGroups(): Create, update, and delete special access groups.
- encryptArticle(), decryptArticle(): Encrypts and decrypts sensitive article bodies.
- logout(): Logs the admin out of the system.

Description: Admins are responsible for managing users, articles, groups, and backups, with additional encryption and special access group management functionalities.

4. ArticleManager Class

Attributes:

- articleList: Stores a list of all articles.

Methods:

- addArticle(): Adds a new article to the system.
- updateArticle(): Updates an existing article.
- deleteArticle(): Deletes an article.
- findArticleById(): Finds an article using its unique ID.
- encryptArticle(), decryptArticle(): Encrypts and decrypts article content.

Description: Handles all operations related to article management, including encryption for sensitive content.

5. BackupHandler Class

Attributes:

- backupFilePath: File path for backups.

Methods:

- backupArticles(): Backs up articles to an external file.
- restoreArticles(): Restores articles from a backup.
- mergeArticles(): Merges restored articles with existing ones.
- backupSpecialAccessGroups(): Backs up special access groups and their content.
- restoreSpecialAccessGroups(): Restores special access groups and their content.

Description: Responsible for securely backing up and restoring articles, groups, and special access content while maintaining data integrity.

6. GroupManager Class

Attributes:

- groupList: Stores a list of all groups.

Methods:

- createGroup(): Creates a new article group.
- deleteGroup(): Deletes an article group.
- addArticleToGroup(): Assigns an article to a specific group.
- removeArticleFromGroup(): Removes an article from a group.
- manageSpecialAccessGroups(): Manages memberships and permissions for special access groups.

Description: Organizes articles into groups for easier categorization and management, including special access groups with restricted permissions.

7. SpecialAccessGroupManager Class

Attributes:

- accessGroupList: Stores a list of special access groups.
- groupPermissions: Tracks permissions for each group.

Methods:

- createSpecialAccessGroup(): Creates a new special access group.
- deleteSpecialAccessGroup(): Deletes a special access group.

- addUserToGroup(): Adds a user to a special access group.
- removeUserFromGroup(): Removes a user from a special access group.
- listGroupMembers(): Lists all members of a special access group.

Description: Introduced to handle the creation, deletion, and management of special access groups and their permissions.

8. EncryptionManager Class

Attributes:

- encryptionKey: The key used for encryption and decryption.

Methods:

- encryptContent(): Encrypts sensitive content.
- decryptContent(): Decrypts sensitive content.
- generateEncryptionKey(): Generates a secure encryption key.

Description: Introduced for handling encryption and decryption of sensitive article content.

9. SearchManager Class

Attributes:

- searchResults: Stores results of the most recent search.

Methods:

- performSearch(): Performs keyword-based searches.
- filterResultsByLevel(): Filters search results by content level.
- filterResultsByGroup(): Filters search results by group.

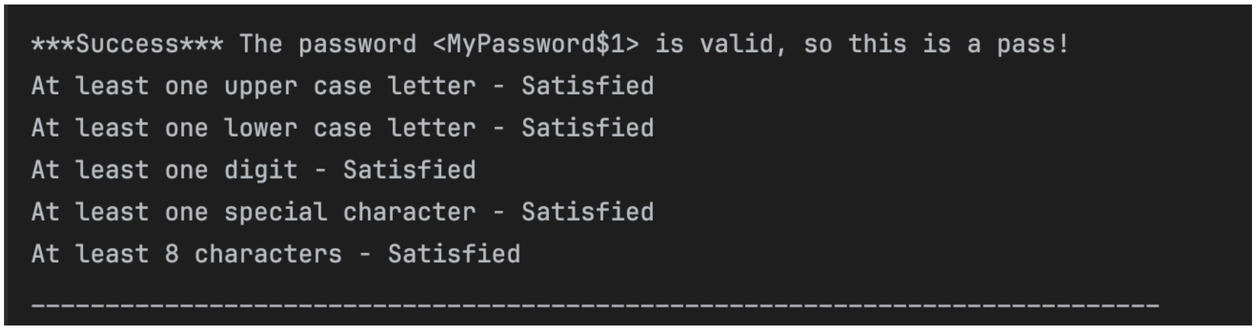
Description: Handles advanced searching functionalities, allowing filtering by level or group introduced.

Class Diagrams: Phase Four

Phase Four focused on ensuring that all requirements and previous feedback from graders were implemented. After a thorough check and testing of requirements, since we received no feedback from graders and got a full score for all the previous phases no change was required to our project.

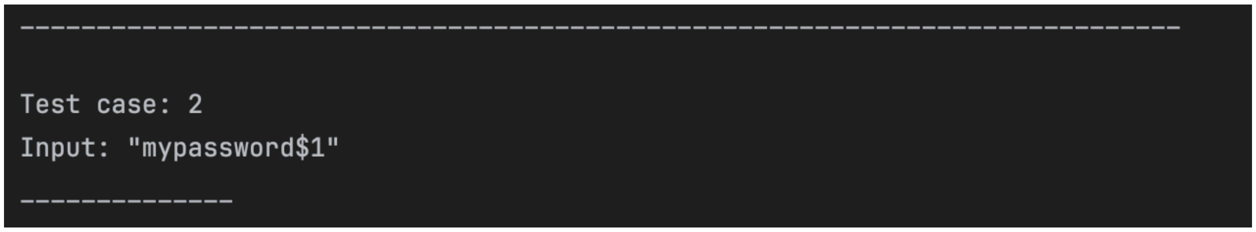
Testing:

Password Testing

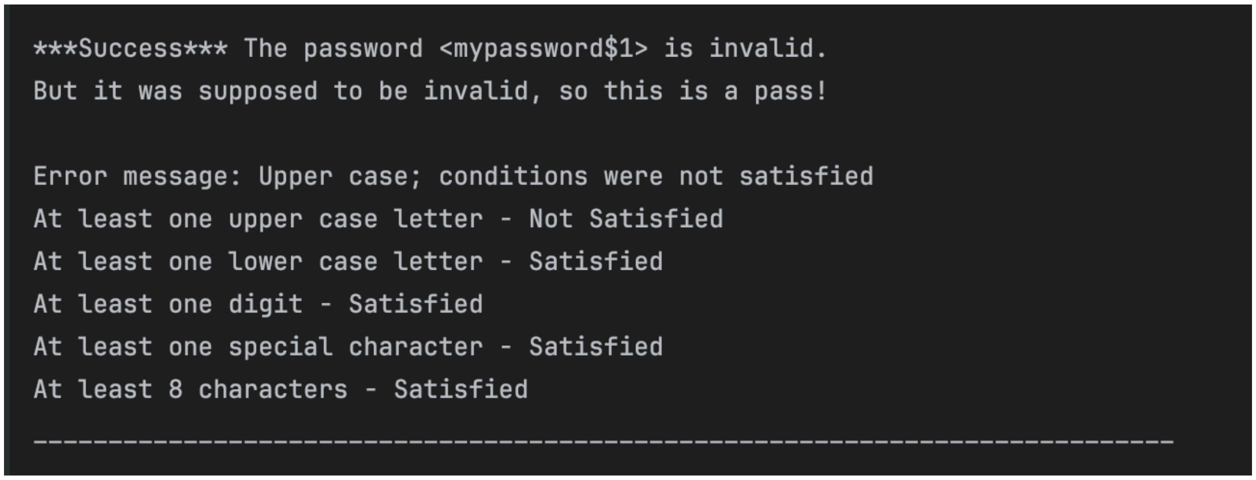


```
***Success*** The password <MyPassword$1> is valid, so this is a pass!
At least one upper case letter - Satisfied
At least one lower case letter - Satisfied
At least one digit - Satisfied
At least one special character - Satisfied
At least 8 characters - Satisfied
----------------------------------------------------------------------------
```

In this test case we entered the correct password that meets all the requirements and we can see that is being shown on terminal testing output as well.

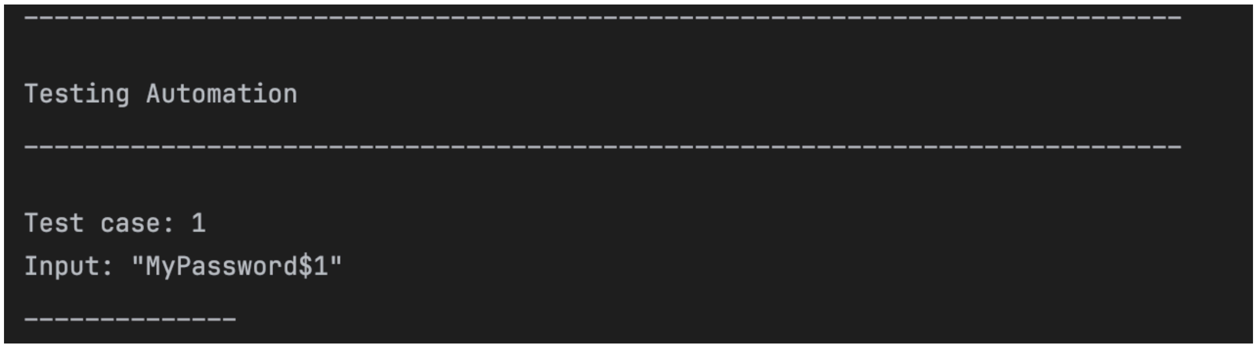


```
------------------------------------------------------------------------------

Test case: 2
Input: "mypassword$1"
--------------
```

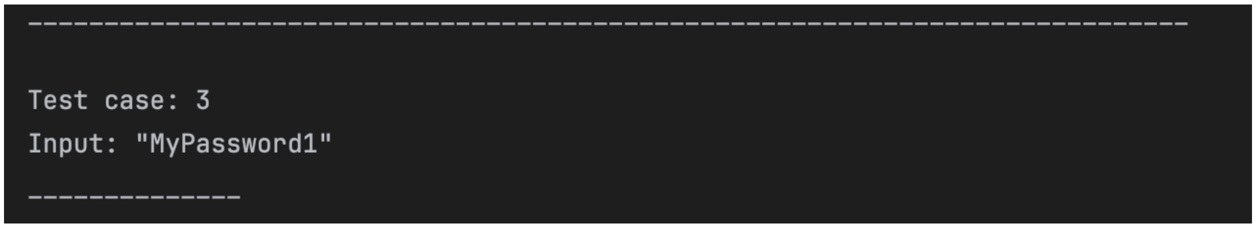


```
***Success*** The password <mypassword$1> is invalid.
But it was supposed to be invalid, so this is a pass!

Error message: Upper case; conditions were not satisfied
At least one upper case letter - Not Satisfied
At least one lower case letter - Satisfied
At least one digit - Satisfied
At least one special character - Satisfied
At least 8 characters - Satisfied
-----------------------------------------------------------------------------
```

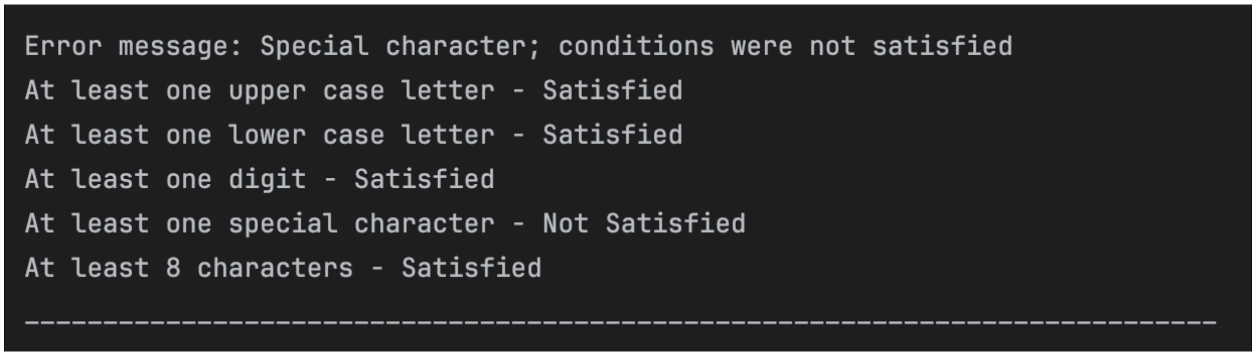
In this test case there is no uppercase character in our password and the condition is not met as seen the terminal output



```
--------------------------------------------------------------------------

Testing Automation

--------------------------------------------------------------------------

Test case: 1
Input: "MyPassword$1"
--------------
```

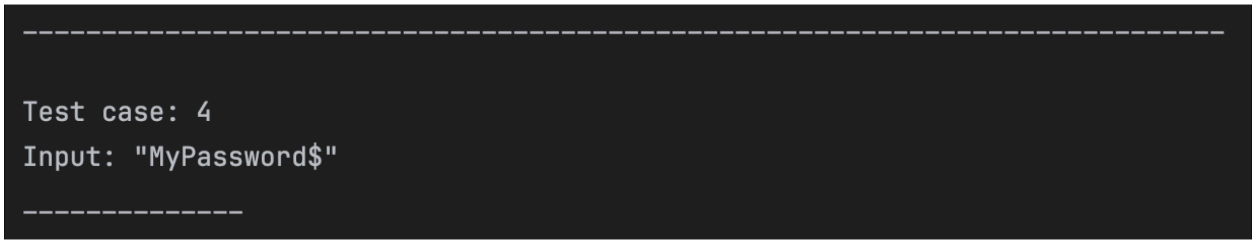


```
--------------------------------------------------------------------------

Test case: 3
Input: "MyPassword1"
--------------
```

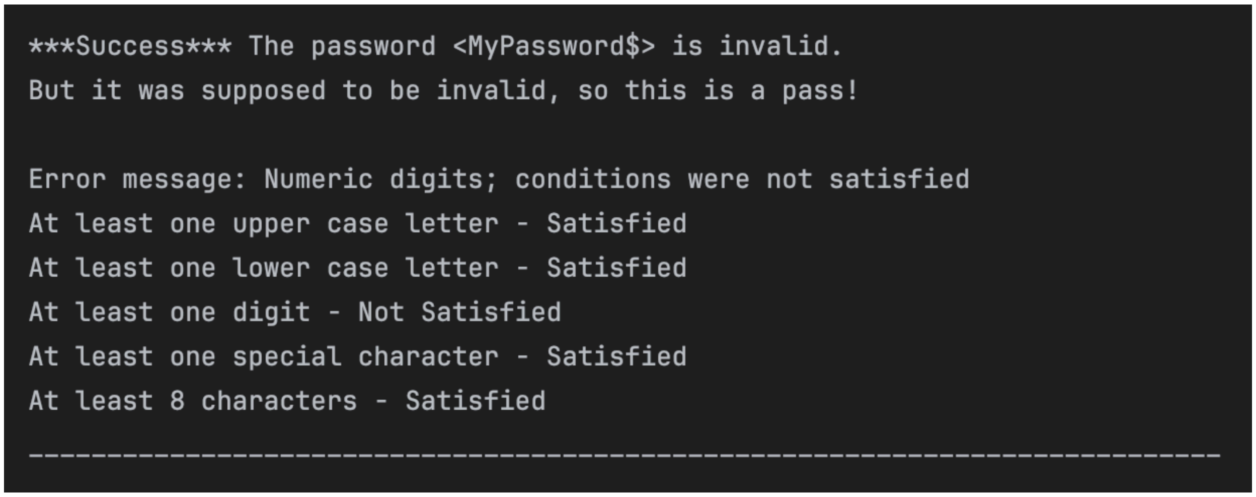


```
Error message: Special character; conditions were not satisfied
At least one upper case letter - Satisfied
At least one lower case letter - Satisfied
At least one digit - Satisfied
At least one special character - Not Satisfied
At least 8 characters - Satisfied
----------------------------------------------------------------------------
```

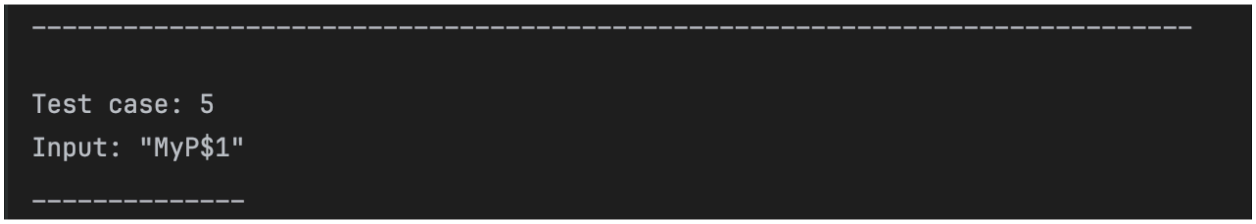
In this test case there is no special character and the condition is not met as shown in the terminal output as well.

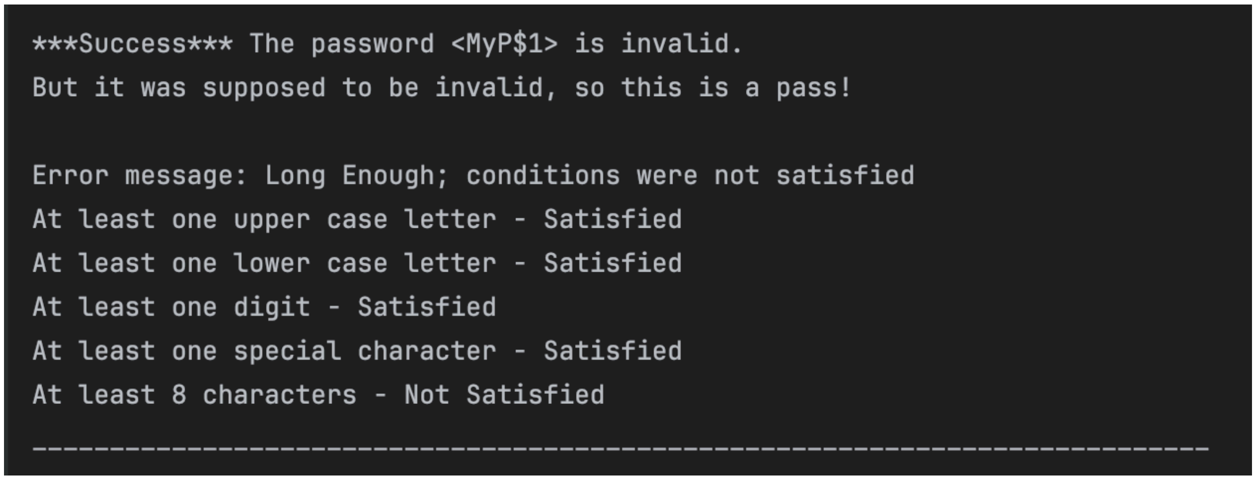


```
----------------------------------------------------------------------------

Test case: 4
Input: "MyPassword$"
--------------
```

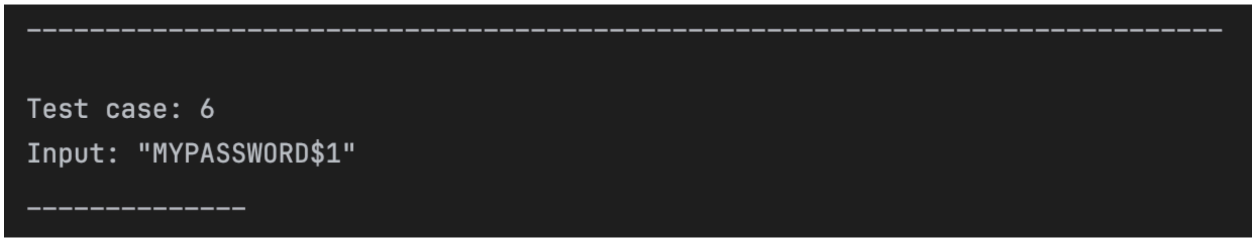

```
***Success*** The password <MyPassword$> is invalid.
But it was supposed to be invalid, so this is a pass!

Error message: Numeric digits; conditions were not satisfied
At least one upper case letter - Satisfied
At least one lower case letter - Satisfied
At least one digit - Not Satisfied
At least one special character - Satisfied
At least 8 characters - Satisfied
----------------------------------------------------------------------------
```

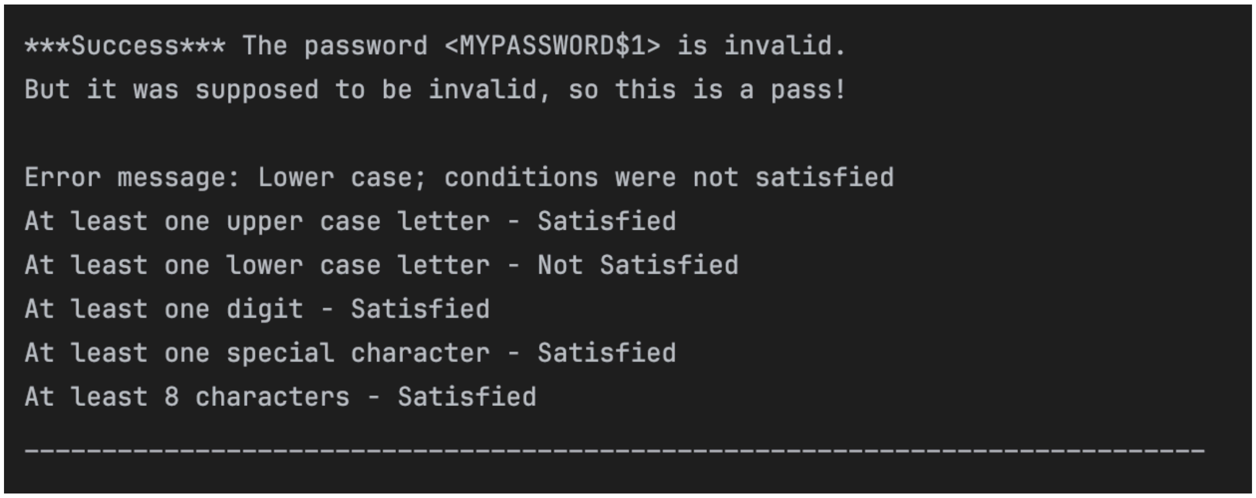
In this test case there is no numeric character and the condition is not met as shown in the terminal output as well.

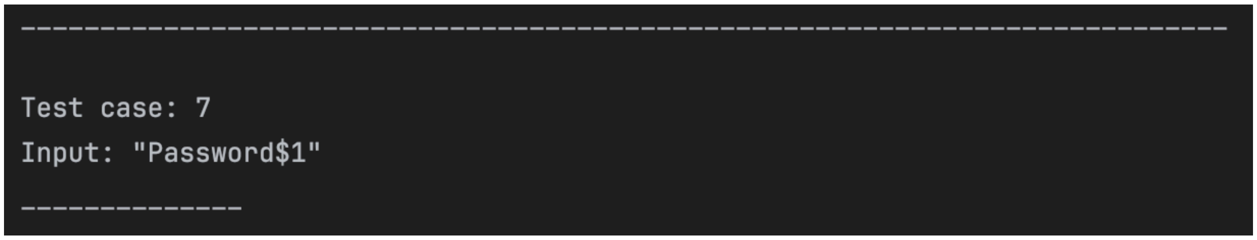
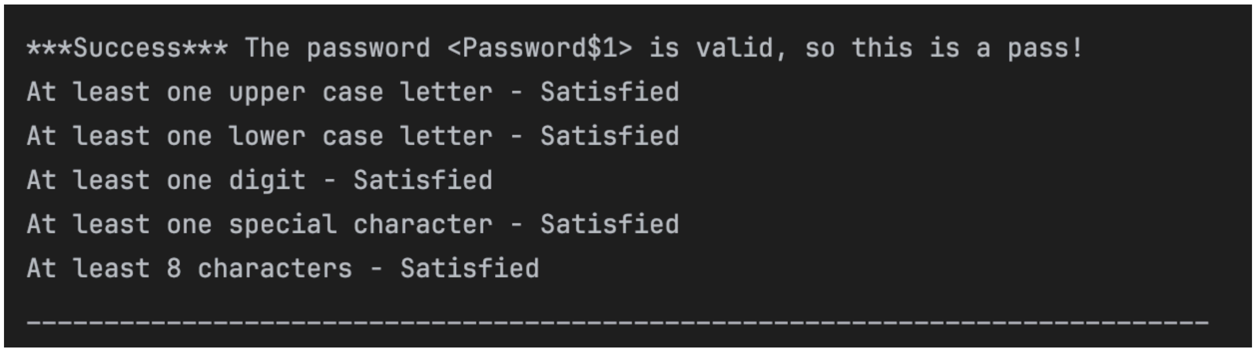

```
--------------------------------------------------------------------------

Test case: 5
Input: "MyP$1"
--------------
```


```
***Success*** The password <MyP$1> is invalid.
But it was supposed to be invalid, so this is a pass!

Error message: Long Enough; conditions were not satisfied
At least one upper case letter - Satisfied
At least one lower case letter - Satisfied
At least one digit - Satisfied
At least one special character - Satisfied
At least 8 characters - Not Satisfied
---------------------------------------------------------------------------
```

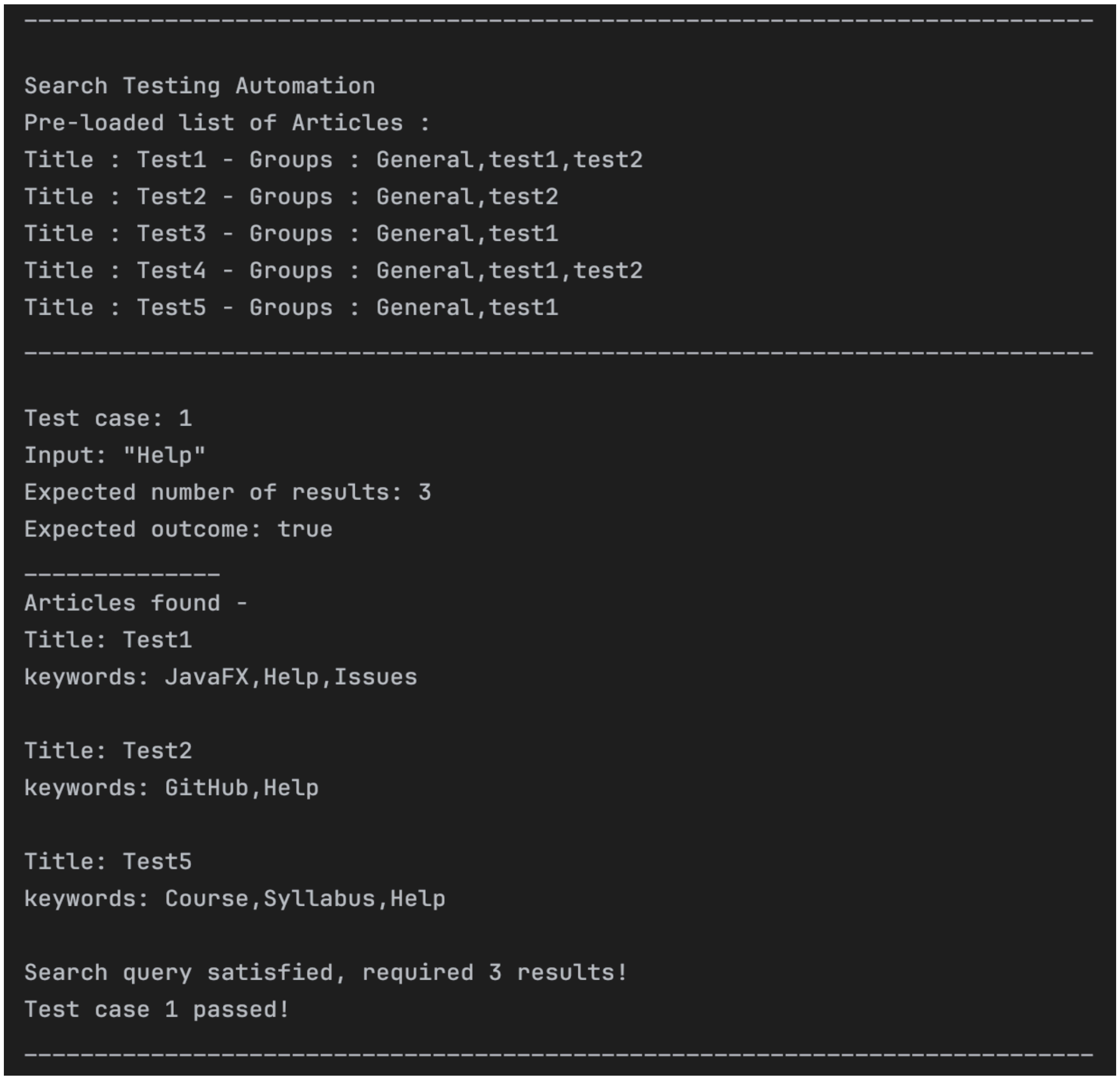
In this test case the password is not long enough and the condition is not met as shown in the terminal output as well.


```
----------------------------------------------------------------------------

Test case: 6
Input: "MYPASSWORD$1"
--------------
```

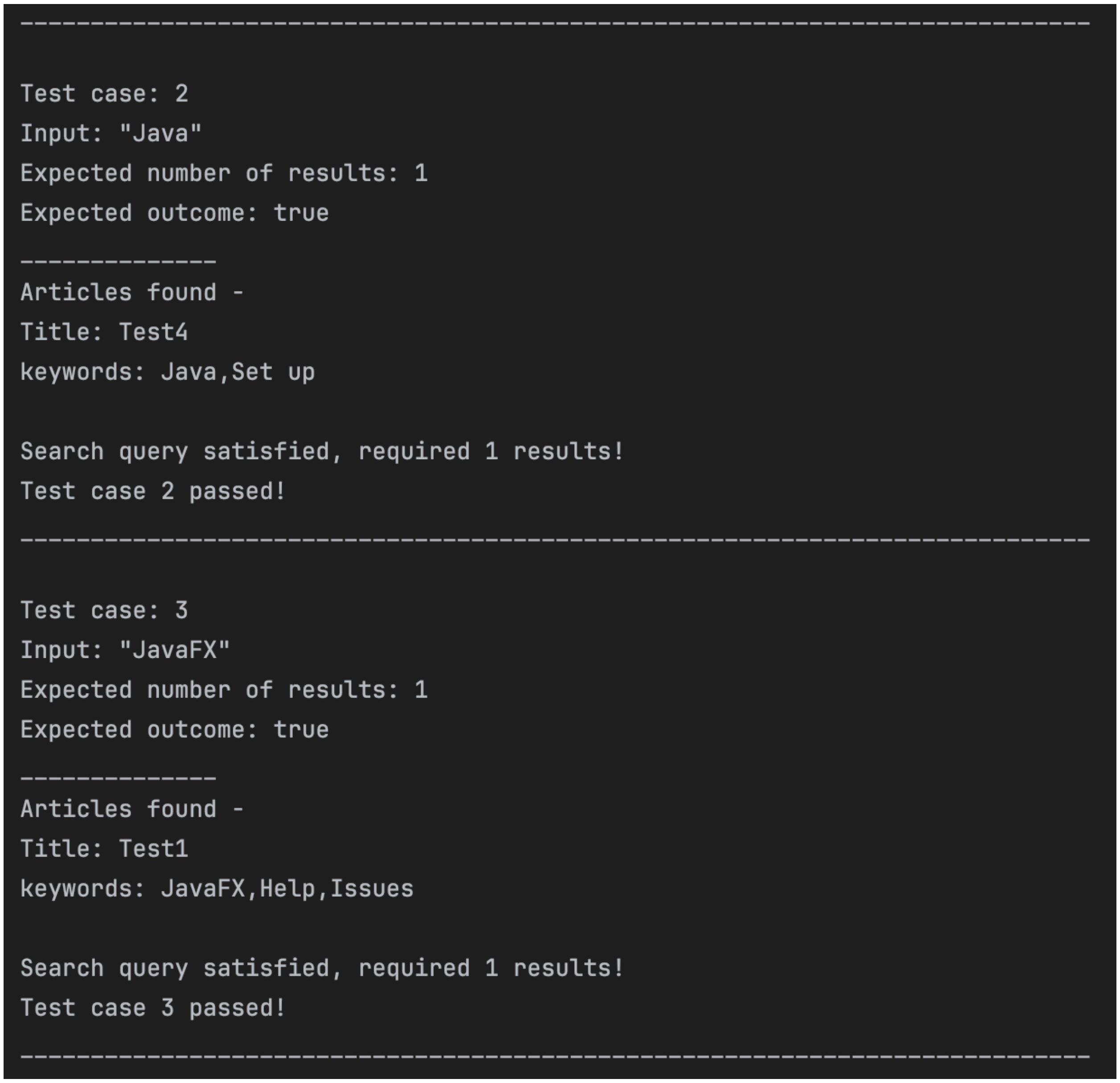

```
***Success*** The password <MYPASSWORD$1> is invalid.
But it was supposed to be invalid, so this is a pass!

Error message: Lower case; conditions were not satisfied
At least one upper case letter - Satisfied
At least one lower case letter - Not Satisfied
At least one digit - Satisfied
At least one special character - Satisfied
At least 8 characters - Satisfied
----------------------------------------------------------------------------
```
In this test case there is no lowercase character and the condition is not met as shown in the terminal output as well.


```
------------------------------------------------------------------------------

Test case: 7
Input: "Password$1"
--------------
```

```
***Success*** The password <Password$1> is valid, so this is a pass!
At least one upper case letter - Satisfied
At least one lower case letter - Satisfied
At least one digit - Satisfied
At least one special character - Satisfied
At least 8 characters - Satisfied
----------------------------------------------------------------------------
```
In this test case we entered the correct password that meets all the requirements and we can see that is being shown on terminal testing output as well.

Search Testing

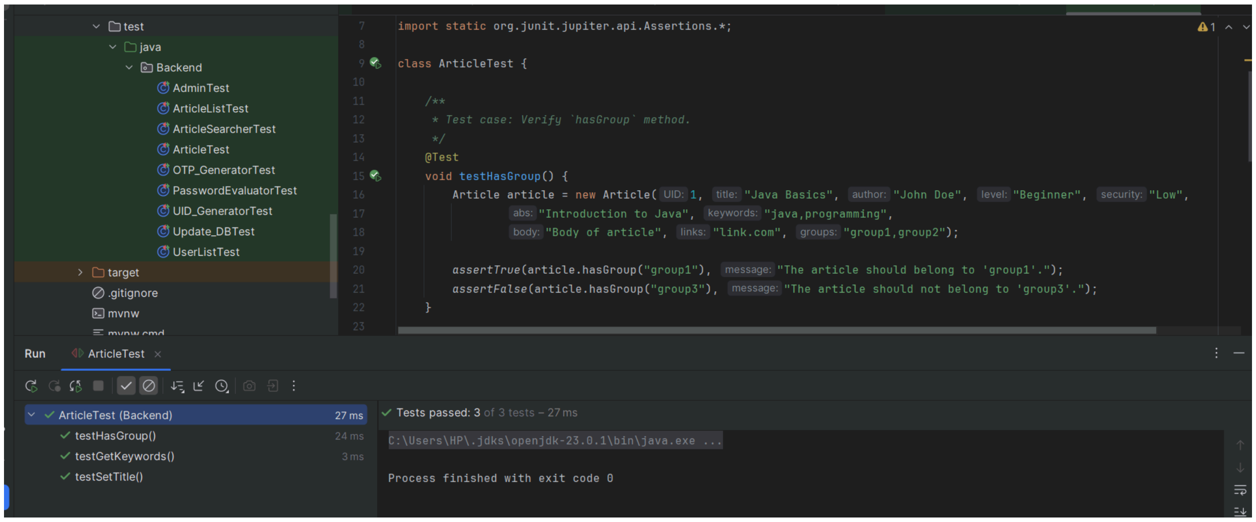


```
----------------------------------------------------------------------------

Search Testing Automation
Pre-loaded list of Articles :
Title : Test1 - Groups : General,test1,test2
Title : Test2 - Groups : General,test2
Title : Test3 - Groups : General,test1
Title : Test4 - Groups : General,test1,test2
Title : Test5 - Groups : General,test1
----------------------------------------------------------------------------

Test case: 1
Input: "Help"
Expected number of results: 3
Expected outcome: true
--------------
Articles found -
Title: Test1
keywords: JavaFX,Help,Issues

Title: Test2
keywords: GitHub,Help

Title: Test5
keywords: Course,Syllabus,Help

Search query satisfied, required 3 results!
Test case 1 passed!
----------------------------------------------------------------------------
```



```
---------------------------------------------------------------------------

Test case: 2
Input: "Java"
Expected number of results: 1
Expected outcome: true
--------------
Articles found -
Title: Test4
keywords: Java,Set up

Search query satisfied, required 1 results!
Test case 2 passed!
---------------------------------------------------------------------------

Test case: 3
Input: "JavaFX"
Expected number of results: 1
Expected outcome: true
--------------
Articles found -
Title: Test1
keywords: JavaFX,Help,Issues

Search query satisfied, required 1 results!
Test case 3 passed!
---------------------------------------------------------------------------
```

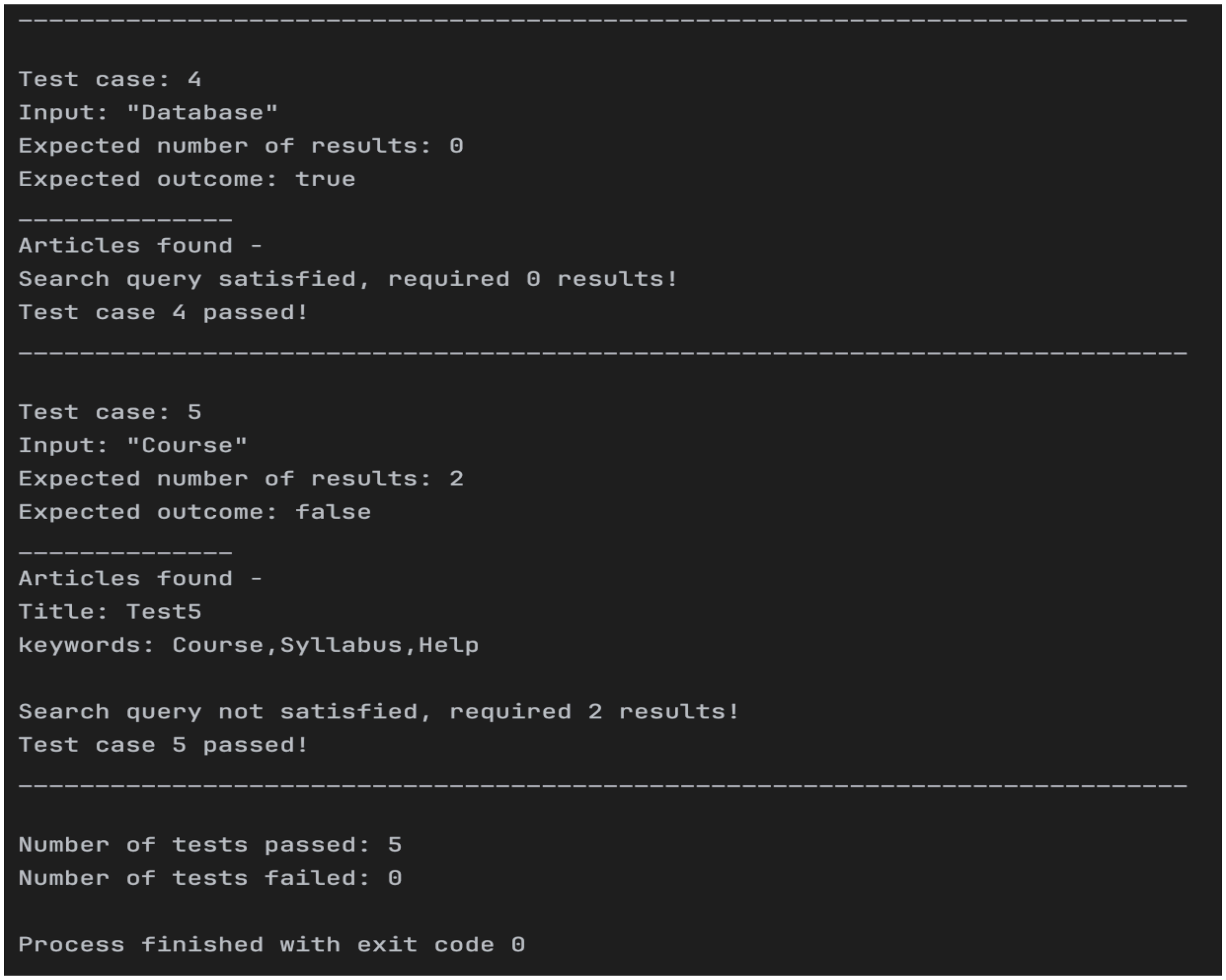
Junit Test Case



```java
import static org.junit.jupiter.api.Assertions.*;

class ArticleTest {

    /**
     * Test case: Verify `hasGroup` method.
     */
    @Test
    void testHasGroup() {
        Article article = new Article( UID: 1,  title: "Java Basics",  author: "John Doe",  level: "Beginner",  security: "Low",
                 abs: "Introduction to Java",  keywords: "java,programming",
                 body: "Body of article",  links: "link.com",  groups: "group1,group2");

        assertTrue(article.hasGroup("group1"),  message: "The article should belong to 'group1'.");
        assertFalse(article.hasGroup("group3"),  message: "The article should not belong to 'group3'.");
    }
```

```
ArticleTest (Backend)   27 ms
  testHasGroup()        24 ms
  testGetKeywords()      3 ms
  testSetTitle()

Tests passed: 3 of 3 tests – 27 ms

C:\Users\HP\.jdks\openjdk-23.0.1\bin\java.exe ...

Process finished with exit code 0
```

3) Article List

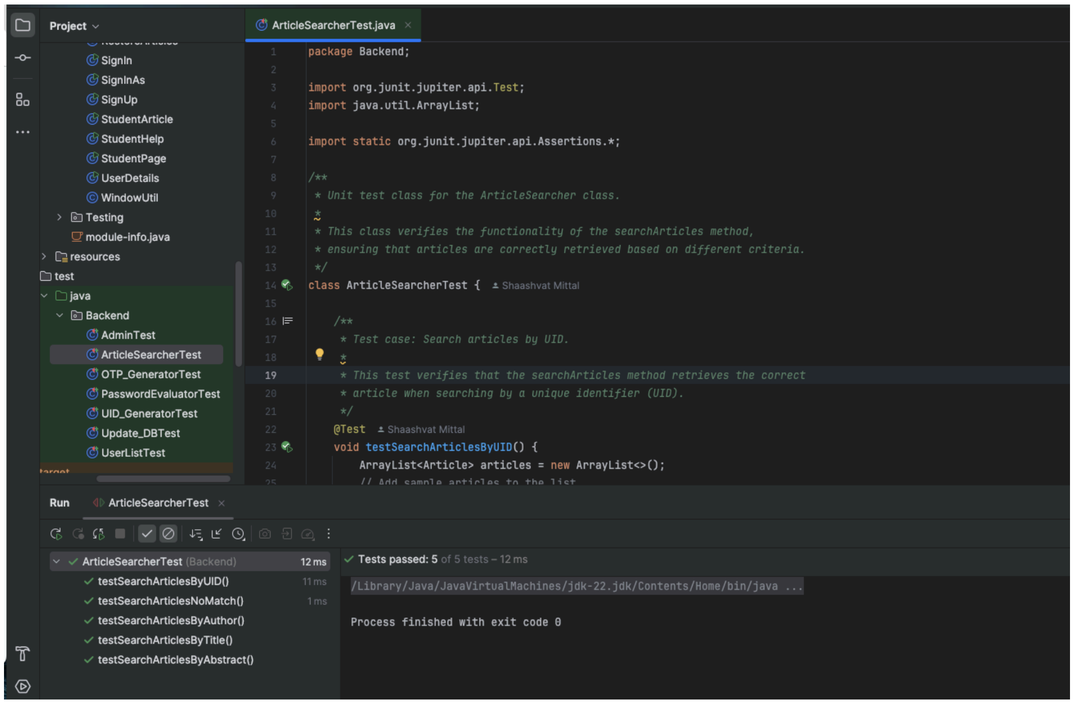


```
----------------------------------------------------------------------------

Test case: 4
Input: "Database"
Expected number of results: 0
Expected outcome: true

--------------
Articles found -
Search query satisfied, required 0 results!
Test case 4 passed!

----------------------------------------------------------------------------

Test case: 5
Input: "Course"
Expected number of results: 2
Expected outcome: false

--------------
Articles found -
Title: Test5
keywords: Course,Syllabus,Help

Search query not satisfied, required 2 results!
Test case 5 passed!

----------------------------------------------------------------------------

Number of tests passed: 5
Number of tests failed: 0

Process finished with exit code 0
```

JUnit Testing:

(The entire JUnit test cases has been added to GitHub as well)

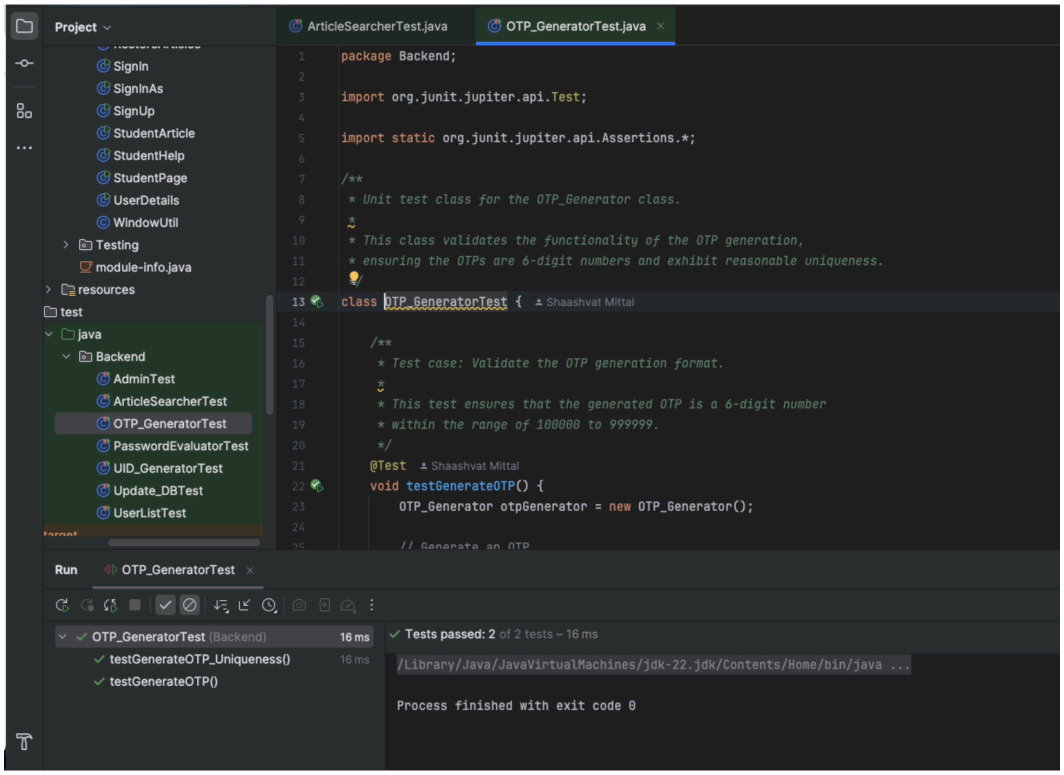
Testing Search:



```java
package Backend;

import org.junit.jupiter.api.Test;
import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Unit test class for the ArticleSearcher class.
 *
 * This class verifies the functionality of the searchArticles method,
 * ensuring that articles are correctly retrieved based on different criteria.
 */
class ArticleSearcherTest {  Shaashvat Mittal

    /**
     * Test case: Search articles by UID.
     *
     * This test verifies that the searchArticles method retrieves the correct
     * article when searching by a unique identifier (UID).
     */
    @Test  Shaashvat Mittal
    void testSearchArticlesByUID() {
        ArrayList<Article> articles = new ArrayList<>();
        // Add sample articles to the list
```

```
✓ ArticleSearcherTest (Backend)      12 ms
    ✓ testSearchArticlesByUID()      11 ms
    ✓ testSearchArticlesNoMatch()     1 ms
    ✓ testSearchArticlesByAuthor()
    ✓ testSearchArticlesByTitle()
    ✓ testSearchArticlesByAbstract()

✓ Tests passed: 5 of 5 tests – 12 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

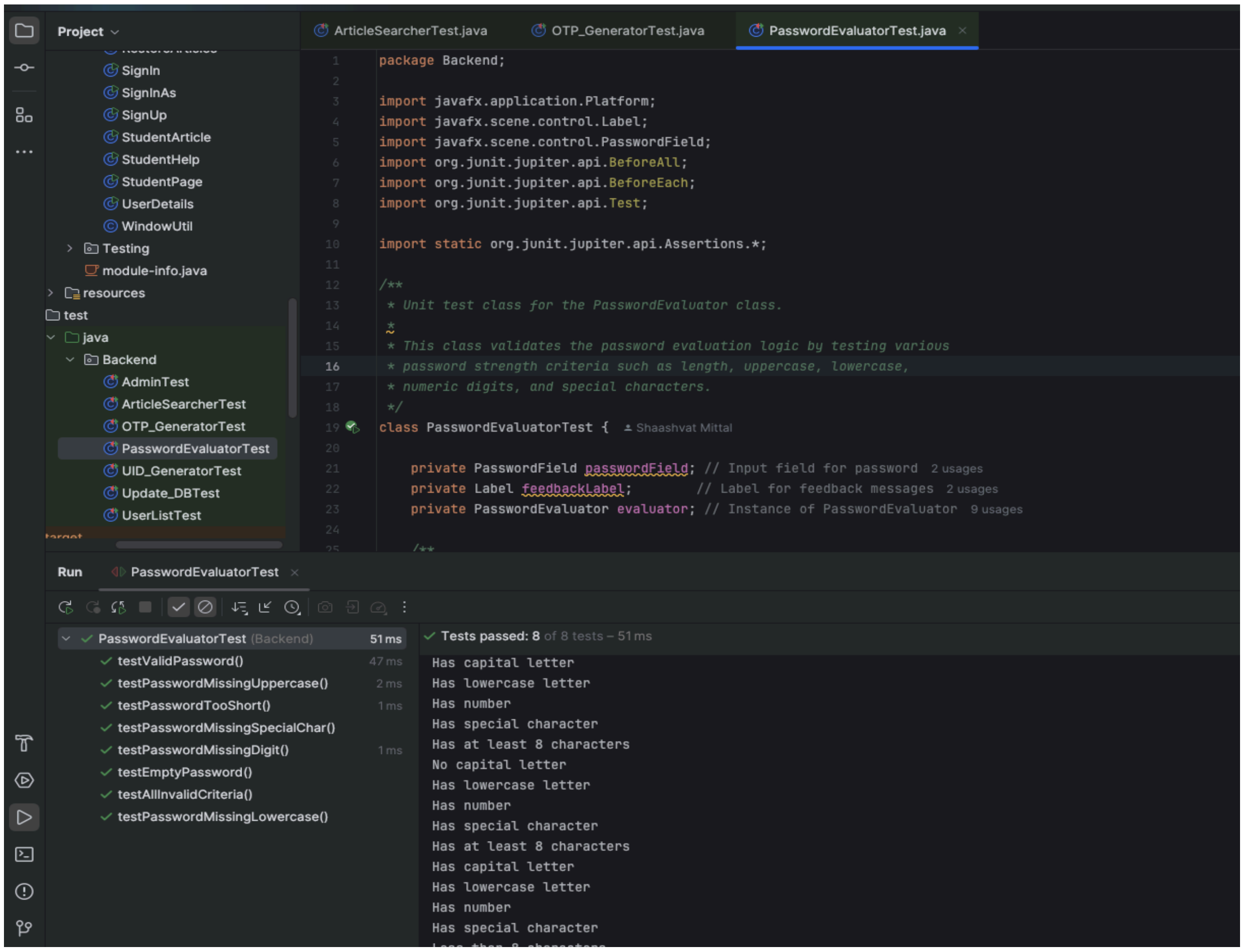
Testing OTP Generation:



```java
package Backend;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Unit test class for the OTP_Generator class.
 *
 * This class validates the functionality of the OTP generation,
 * ensuring the OTPs are 6-digit numbers and exhibit reasonable uniqueness.
 */
class OTP_GeneratorTest {  Shaashvat Mittal

    /**
     * Test case: Validate the OTP generation format.
     *
     * This test ensures that the generated OTP is a 6-digit number
     * within the range of 100000 to 999999.
     */
    @Test  Shaashvat Mittal
    void testGenerateOTP() {
        OTP_Generator otpGenerator = new OTP_Generator();

        // Generate an OTP
```

```
OTP_GeneratorTest (Backend)  16 ms
  testGenerateOTP_Uniqueness()  16 ms
  testGenerateOTP()

Tests passed: 2 of 2 tests – 16 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Testing Password Evaluator:

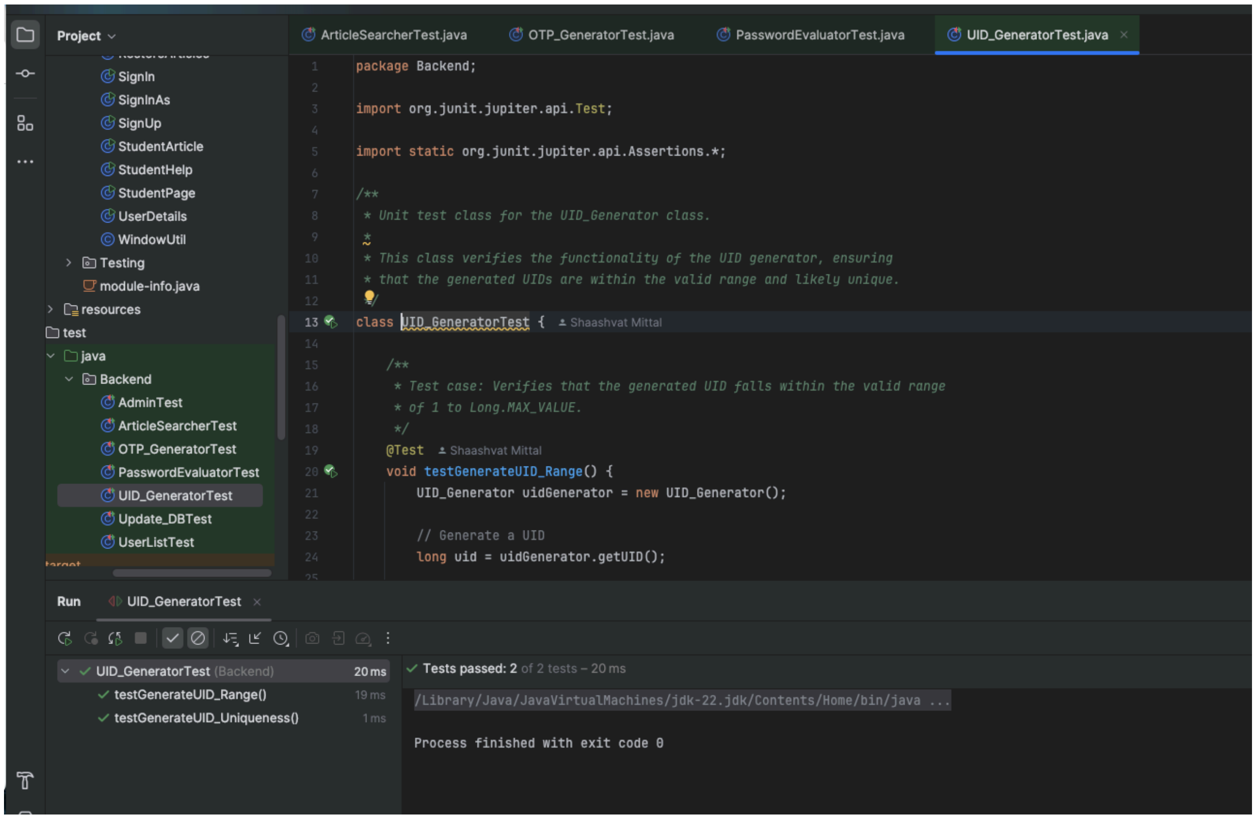


```java
package Backend;

import javafx.application.Platform;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Unit test class for the PasswordEvaluator class.
 *
 * This class validates the password evaluation logic by testing various
 * password strength criteria such as length, uppercase, lowercase,
 * numeric digits, and special characters.
 */
class PasswordEvaluatorTest {  Shaashvat Mittal

    private PasswordField passwordField; // Input field for password  2 usages
    private Label feedbackLabel;         // Label for feedback messages  2 usages
    private PasswordEvaluator evaluator; // Instance of PasswordEvaluator  9 usages

    /**
```

```
PasswordEvaluatorTest (Backend)   51 ms
  testValidPassword()              47 ms
  testPasswordMissingUppercase()    2 ms
  testPasswordTooShort()            1 ms
  testPasswordMissingSpecialChar()
  testPasswordMissingDigit()        1 ms
  testEmptyPassword()
  testAllInvalidCriteria()
  testPasswordMissingLowercase()
```

```
Tests passed: 8 of 8 tests – 51 ms
Has capital letter
Has lowercase letter
Has number
Has special character
Has at least 8 characters
No capital letter
Has lowercase letter
Has number
Has special character
Has at least 8 characters
Has capital letter
Has lowercase letter
Has number
Has special character
```

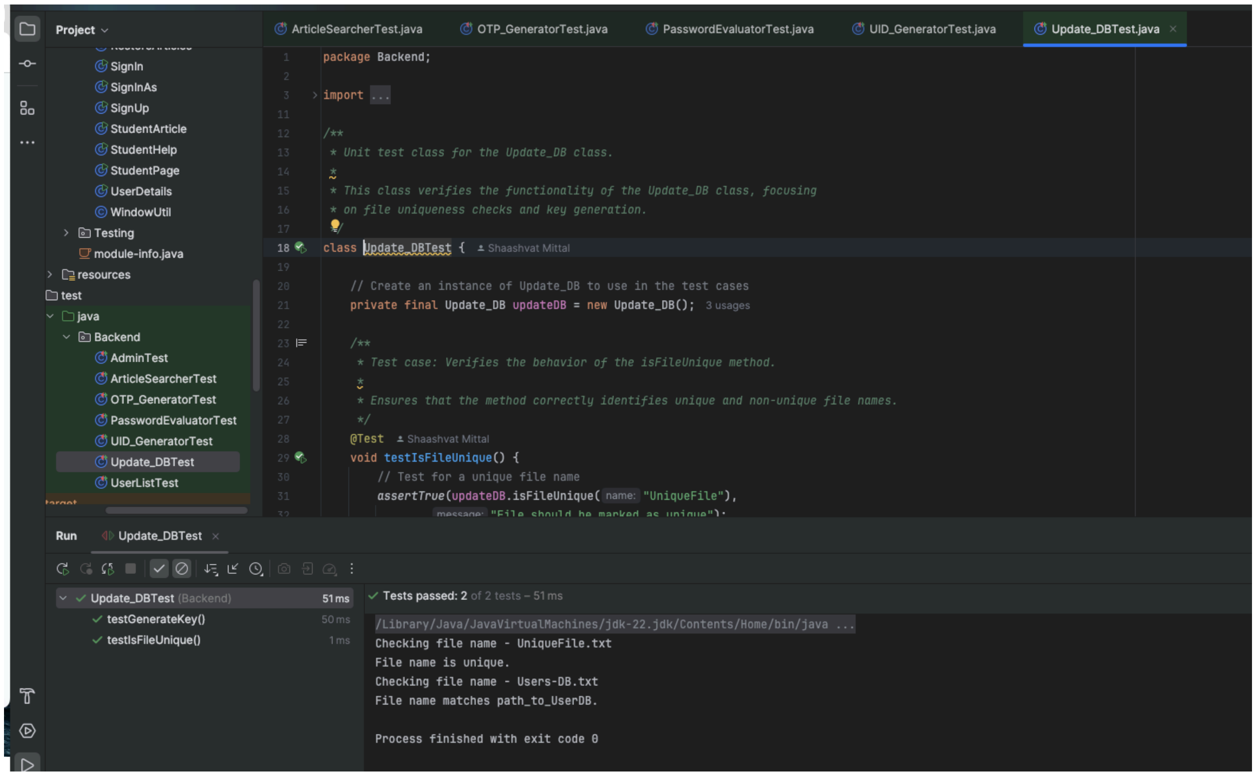
Testing Unique ID Generator:



```java
package Backend;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Unit test class for the UID_Generator class.
 *
 * This class verifies the functionality of the UID generator, ensuring
 * that the generated UIDs are within the valid range and likely unique.
 */
class UID_GeneratorTest {  Shaashvat Mittal

    /**
     * Test case: Verifies that the generated UID falls within the valid range
     * of 1 to Long.MAX_VALUE.
     */
    @Test  Shaashvat Mittal
    void testGenerateUID_Range() {
        UID_Generator uidGenerator = new UID_Generator();

        // Generate a UID
        long uid = uidGenerator.getUID();
```

```
✓ UID_GeneratorTest (Backend)        20 ms
    ✓ testGenerateUID_Range()        19 ms
    ✓ testGenerateUID_Uniqueness()    1 ms

✓ Tests passed: 2 of 2 tests – 20 ms
/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

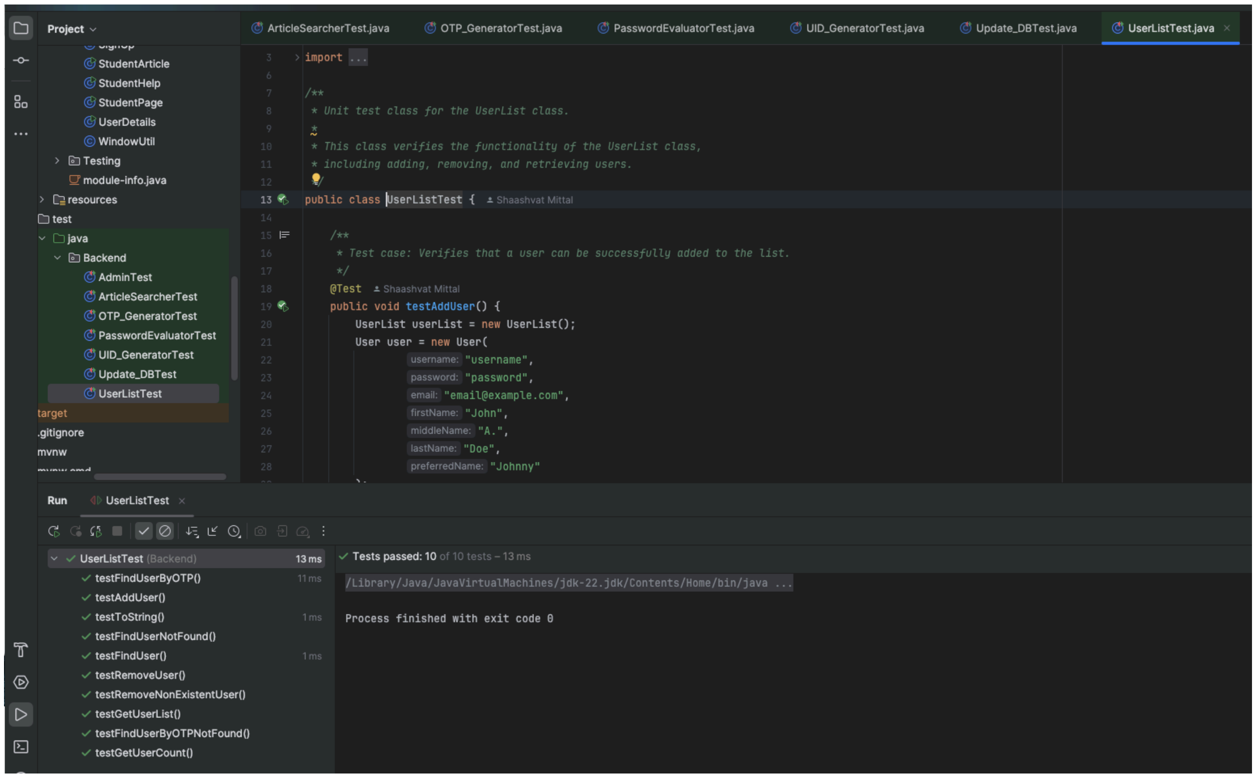
Testing Update_DB:



```java
package Backend;

import ...

/**
 * Unit test class for the Update_DB class.
 *
 * This class verifies the functionality of the Update_DB class, focusing
 * on file uniqueness checks and key generation.
 */
class Update_DBTest {  Shaashvat Mittal

    // Create an instance of Update_DB to use in the test cases
    private final Update_DB updateDB = new Update_DB();  3 usages

    /**
     * Test case: Verifies the behavior of the isFileUnique method.
     *
     * Ensures that the method correctly identifies unique and non-unique file names.
     */
    @Test  Shaashvat Mittal
    void testIsFileUnique() {
        // Test for a unique file name
        assertTrue(updateDB.isFileUnique( name: "UniqueFile"),
```

```
✓ Update_DBTest (Backend)     51 ms
    ✓ testGenerateKey()       50 ms
    ✓ testIsFileUnique()       1 ms

✓ Tests passed: 2 of 2 tests – 51 ms
/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...
Checking file name - UniqueFile.txt
File name is unique.
Checking file name - Users-DB.txt
File name matches path_to_UserDB.

Process finished with exit code 0
```

Testing User List:



```java
import ...

/**
 * Unit test class for the UserList class.
 *
 * This class verifies the functionality of the UserList class,
 * including adding, removing, and retrieving users.
 */
public class UserListTest {  Shaashvat Mittal

    /**
     * Test case: Verifies that a user can be successfully added to the list.
     */
    @Test  Shaashvat Mittal
    public void testAddUser() {
        UserList userList = new UserList();
        User user = new User(
                username: "username",
                password: "password",
                email: "email@example.com",
                firstName: "John",
                middleName: "A.",
                lastName: "Doe",
                preferredName: "Johnny"
```

```
UserListTest (Backend)              13 ms
  testFindUserByOTP()               11 ms
  testAddUser()
  testToString()                     1 ms
  testFindUserNotFound()
  testFindUser()                     1 ms
  testRemoveUser()
  testRemoveNonExistentUser()
  testGetUserList()
  testFindUserByOTPNotFound()
  testGetUserCount()
```

```
Tests passed: 10 of 10 tests – 13 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Screencasts

Phase 1:

How-to-use Screencast:
https://asu.zoom.us/rec/share/dq8pqQ2kDyKLkAZBB6eNfZLEFbGt2Lk_rcHoCSU6A_DdLOiydhCzVJL4ZX8hXtbt.2elsIuIF1-84Wd__?startTime=1728517711000
Passcode: Y!cYzZ&4

Technical Screencast:
https://asu.zoom.us/rec/share/QZs_44qox1aPmxS7bIHasQAHRZI6Pp0JqzMVtx6u92a0jChHBS_F6EkBQFuflMvx.Lz-BjYptogfq0j-j
Passcode: P@Yj3#@h

Phase 2:

How-to-use screencast:
https://asu.zoom.us/rec/share/JSraHZi6K1Txr48iAWlB5N4y-onH3Iheo5fZmW4fAUu-FPkTzUdXE-YDfVplMx6m._fRKeds-VohpjoUH?startTime=1730277470000
Passcode: !EY7Qak$

Technical Screencast:
https://asu.zoom.us/rec/share/pVmnzfYEpf97NIyD1HU3SdiEgCsQdnglBFHPOAr9uIlFOUL-wqstQ6ywq1JRqQ9w.NoABAyHf9_utoq-z

Phase 3:

How-to-use screencast:
https://asu.zoom.us/rec/share/YKmsxwZ2DDBzEA8ommih_5icxwUlBzcX8L65m3bOUAPXkiP2EHnPynF6V6JYshq_.vCIXCIGipvPY5nRa?startTime=1732127970000
Passcode: %pz1Hba0

Technical Screencast:
Code walkthrough
JUnit test cases walkthrough

Github Link:

https://github.com/shaashvat01/ASU-Help-System

Credit Page

Team Member Name	Contributions
Shaashvat Mittal	Completed assigned testing for the project to ensure all functionalities are covered. Reviewed the document for the final submission.
Mukesh Tulluru	Conducted thorough functionality testing and reviewed the final document for submission.
Naga Sathvik Kommareddy	Performed comprehensive testing of functionalities and finalized the submission document through review.
Saisrivathsan Manikandan	Completed assigned testing for the project to ensure all functionalities are covered. Reviewed the document for the final submission.
Gayathri Kota	Completed rigorous functionality testing and provided a review of the final submission document.