Project 2

Access Control

By Cameron Fisher and Jacob Ilas
Professor Al-Nashif
CIS4367.01
12/5/2022

Task:

1) Create the access control matrix for the system.
2) Implement the access control mechanisms and policies for the system above. Keep in mind that the manuscript's author can also be a reviewer or an associate editor for that manuscript.

Permissions to be granted

Authors

- Submit manuscript
    - Could take on reviewer and associate editor roles
        - Did not grant same permissions as above roles because they *can* also be these roles per the project description which implies it's possible but not guaranteed

Editors

- Invite Associate/Guest Editors
    - manage review process
    - quality control review process

Associate Editors

- Ask reviewers to review (invite)
- Review manuscript
- Make recommendation

Reviewers

- Submit review
    - Scores
    - Recommendations

Administrators

- - ???
- - Created printing ACM as permissions were not explicitly stated

Since it was stated in class that the expectation is simply an implementation of the ACM, there is no full functionality of the manuscript system. All that remains are checks for access to certain functions.

Environment

The programming language we used was Rust and we used VSCode to code the project, the following link provides a guide to configuring the CLI environment.

https://www.rust-lang.org/learn/get-started

You need to be in the proper directory where the src folder is located, or else cargo will not be able to run properly.
e.g.,

```
C:\Users\Jacob\Downloads\AccessControlProject-main\AccessControlProject-main>
```

*** When compiling the ACL Project, under login.rs you need to specify the correct path for the csv files to be read, or else the code will fail. ***

```
// CSV Reader reads csv file of users
let mut reader = csv::Reader::from_path("C:/Users/Jacob/Desktop/users.csv").unwrap();
let result = reader.records();
```

```
// CSV Reader reads csv file of users
let mut reader = csv::Reader::from_path("C:/Users/Jacob/Desktop/admin.csv").unwrap();
let result = reader.records();
```

By default the expected file structure of the project is:

```
|---{Program Root Directory}.
|   admin.csv
|   Cargo.lock
|   Cargo.toml
|   users.csv
|
\---src
        acm.rs
        login.rs
        main.rs
```

Expected format of csv files:

users.csv
First_Name, Last_Name, Username, Password, Role

Admin.csv
First_Name, Last_Name, Username, Password

Expected “Role” Values:
Author, Editor, Associate Editor, Reviewer
***** The code handles case sensitivity but the roles must match the exact spelling *****

Access Control Matrix

The access control matrix is a simple 4 row, 5 column 2d matrix that stores boolean values to denote if a role can access the function attempted.

Admin print ACM functionality

```
Username:
admin1
Password:
password
1. Print Access Control Matrix
2. Close program
1
_____________________________________________________________________________________________________
|       Role       | Submit Manuscript | Invite Associate Editor | Invite Reviewer | Submit Review | Make Recommendation |
_____________________________________________________________________________________________________
|      Author      |       true        |          false          |      false      |     false     |        false        |
_____________________________________________________________________________________________________
|      Editor      |       false       |          true           |      false      |     false     |        false        |
_____________________________________________________________________________________________________
| Associate Editor |       false       |          false          |      true       |     true      |        true         |
_____________________________________________________________________________________________________
|     Reviewer     |       true        |          false          |      false      |     true      |        true         |
_____________________________________________________________________________________________________
```

How ACM is generated in code (reflects in above test case)

```rust
pub struct ACM{
    // Roles Array (Row, 1st num) in order:
    // author, editor, associate_editor, reviewer
    // Actions/Permissions (Column, 2nd num) array in order:
    // submit, invite_associates, invite_reviewers, submit_review, make_recommendation
    // Usage: acm[role_index][permission_index]
    acm: [[bool; 5]; 4],
    roles: HashMap<String, usize>,
}

// Generates ACM for main.rs such that it can be passed to the action functions
pub fn gen_acm() -> ACM {
    // Actions/Permissions array in order:
    // submit, invite_associates, invite_reviewers, submit_review, make_recommendation

    // Author
    let author_actions: [bool; 5] = [true, false, false, false, false];

    // Editor
    let editor_actions: [bool; 5] = [false, true, false, false, false];

    // Associate Editor
    let associate_editor_actions: [bool; 5] = [false, false, true, true, true];

    // Reviewer
    let reviewer_actions: [bool; 5] = [false, false, false, true, true];

    // Compile permissions into 2D matrix
    let ac_matrix: [[bool; 5]; 4] = [author_actions, editor_actions, associate_editor_actions, reviewer_actions];
```

Granting Access

When the user provides login credentials the login.rs file will consult the users.csv file, if the user is located they will be able to select from a list of available actions, if they are not located they will be asked if they want to retry or exit the program.

```
    // Iterate through CSV Results for match
    for record in result {
        let record = record.unwrap();
        // 3rd index (get(2)) and 4th index (get(3)) are username and password respectively.
        // Need to unwrap() to get &str and trim() to pare down to the ASCII characters alone
        if username == String::from(record.get(2).unwrap().trim()) && pass == String::from(record.get(3).unwrap().trim()) {
            logged_in = true;
            role = String::from(record.get(4).unwrap().trim());
            break;
        }
    }

    // Check if user wants to try loggin in again. Mostly kept as a way of leaving without mashing ctrl + c all the time while testing
    if !logged_in {
        println!("Username or password was incorrect! \n Try again? (y/n)");
        let mut selection = String::new();
        std::io::stdin().read_line(&mut selection).unwrap();
        let selection = selection.trim();
        if selection == "n"{
            break;
        }
    }
}
```

In the main.rs file, after selecting a function to attempt, the logged in user's role will be ran against an ACM which will display the permission based on the role the user is assigned.

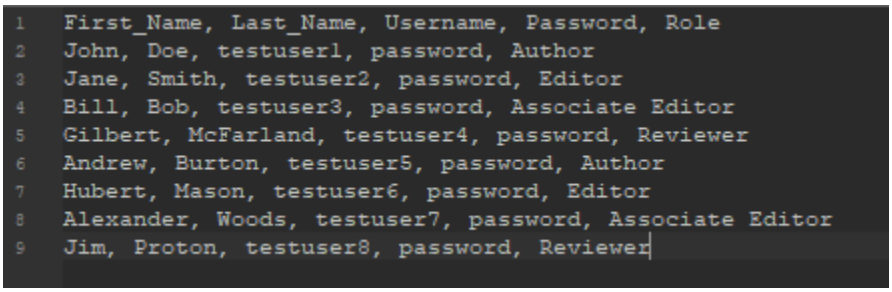
```
    // User Actions Prompt
    loop {
        if user_logged_in != "0" {
            // Create ACM
            // Configure default in acm.rs
            let access_control = acm::gen_acm();
            let role = &user_logged_in.to_ascii_lowercase(); // Send into function for permission cehcking

            // Menu
            println!("1. Submit manuscript");
            println!("2. Invite associate editors");
            println!("3. Invite reviewers");
            println!("4. Submit review");
            println!("5. Make recommendation");
            println!("6. Close program");
            let mut selection = String::new();
            std::io::stdin().read_line(&mut selection);
            let selection = selection.trim();
            match selection {
                "1" => acm::submit_manuscript(access_control, role),
                "2" => acm::invite_editors(access_control, role),
                "3" => acm::invite_reviewers(access_control, role),
                "4" => acm::submit_review(access_control, role),
                "5" => acm::make_recommendation(access_control, role),
                "6" => break,
                _ => invalid_input().unwrap()
            };
        } else {
            break;
        }
    }
```

Executing/Testing Code

```
1. Login as User
2. Login as Admin
3. Register User
4. Close program
1
Loading Users...
--- Login ---
Username:
testuser1
Password:
password
1. Submit manuscript
2. Invite associate editors
3. Invite reviewers
4. Submit review
5. Make recommendation
6. Close program
1
Allowed
1. Submit manuscript
2. Invite associate editors
3. Invite reviewers
4. Submit review
5. Make recommendation
6. Close program
2
Not Allowed
1. Submit manuscript
2. Invite associate editors
3. Invite reviewers
```

Based on the provided credentials above the user is “Allowed” to submit a manuscript but cannot execute any of the other actions due to the permissions provided by their role: “Not Allowed”. This is because the user is an “Author” per the users.csv file



```
First_Name, Last_Name, Username, Password, Role
John, Doe, testuser1, password, Author
Jane, Smith, testuser2, password, Editor
Bill, Bob, testuser3, password, Associate Editor
Gilbert, McFarland, testuser4, password, Reviewer
Andrew, Burton, testuser5, password, Author
Hubert, Mason, testuser6, password, Editor
Alexander, Woods, testuser7, password, Associate Editor
Jim, Proton, testuser8, password, Reviewer
```

Consulting the ACM created shows that authors are only permitted to submit a manuscript.