# SYSC 4810

# Assignment 1 Report

# Arjun Pathak

# 101212581

* 1. RBAC seems to be the most fit. RBAC assigns those permissions based on the role assigned to the user within the enterprise. This fits justInvest's structure, which has specific roles: Clients, Premium Clients, Financial Advisors, Financial Planners, and Tellers. The model simplifies permission administration by grouping permissions according to job functions-so that users will only have access to the information and operations they need.

A white board with red lines and words

Description automatically generated with medium confidence

* 1. Valid User Authorization:

**Valid User Authorization (Client User):**

**Test**: A client user logs in with correct credentials and attempts a permitted operation (e.g., view\_account\_balance).

**Coverage**: Verifies that authorized operations for a valid, basic role (Client) are correctly allowed.

**Unauthorized Operation (Client User):**

**Test**: A client user tries an operation they are not permitted to perform (e.g., modify\_investment\_portfolio).

**Coverage**: Ensures that operations outside the user’s assigned permissions are correctly denied.

**Business Hours Check (Teller Role - Within Business Hours):**

**Test**: A Teller performs an allowed action (e.g., view\_any\_account\_balance) during simulated business hours.

**Coverage**: Confirms that Tellers can access their authorized operations only during the defined operational window.

**Business Hours Check (Teller Role - Outside Business Hours):**

**Test**: A Teller attempts the same operation outside business hours.

**Coverage**: Verifies that Tellers are restricted when it’s not business hours, enforcing time-based policies.

**Premium Client Privileges:**

**Test**: A Premium Client executes both standard client operations and premium-only operations (modify\_investment\_portfolio, view\_planner\_contact).

**Coverage**: Checks that enhanced roles receive their added permissions.

**Invalid Credentials:**

**Test**: An incorrect username/password combination.

**Coverage**: Ensures that unauthorized users cannot gain access.

**Role-Specific Operations (Financial Advisor vs. Financial Planner):**

**Test**: Tests a Financial Advisor’s exclusive permissions (e.g., view\_private\_consumer\_instruments) and denies those not in their permission set (e.g., view\_money\_market\_instruments), and similarly checks differences with a Financial Planner.

**Coverage**: Confirms that higher-level financial roles have distinct sets of operations and that these are enforced correctly.

1. To securely store user passwords, we employ the **Argon2** hashing algorithm.

This hashing algorithm specifically designed to protect credentials

**Key Parameters:**

* **Salt Generation and Length:** A unique 16-byte (128-bit) salt is generated for each password using a secure random number generator .This ensures that identical passwords yield distinct hashes.
* **Hash Length:** The output hash is set to 32 bytes (256 bits), balancing security and storage efficiency.

**Username:** A unique identifier for the user.

**Salt:** A unique, randomly generated value for each user, used to ensure that identical passwords result in distinct hashes.

**Hashed Password:** The result of hashing the user's password combined with the salt using the Argon2id algorithm.

**Client role:** The role of the user who is trying to log in

**Each component is separated by a ‘:’**

**Example:**

arjun**:**cbee32462c0d5285093370e33fcb6e42**:**$argon2id$v=19$m=65536,t=3,p=4$rCnfrgWR2EYrhGwYOjIQIw$GSLX7Z4+RhjCusyAoyK8Qv0OHW0CsByQ3qXUYqOrhmQ**:**client

d.

**Test Adding a New User (test\_add\_new\_user):**

**Test:**

Ensures that when a new user is added, their record is properly written to the password file.

**Coverage**:

This test checks basic file-writing functionality and confirms that the expected format (username:salt:hashed\_password:client) is present. Verifying the correct line structure ensures that the stored data meets the defined specification.

**Test Preventing Duplicate Usernames (test\_prevent\_duplicate\_usernames):**

**Test:**

Attempts to add a user with a username that already exists in the file, expecting a ValueError to be raised.

**Coverage:**

This ensures that the system enforces uniqueness of usernames, preventing overwriting or unintended conflicts in the password file. It covers error handling for duplicate entries and data integrity constraints.

**Test Verifying Correct Password (test\_verify\_correct\_password):**

**Test:**

After adding a user, verifies that using the correct password returns True.

**Coverage:**

This validates that the password verification mechanism works as intended for valid credentials, covering the crucial functionality of authentication.

**Test Verifying Incorrect Password (test\_verify\_incorrect\_password):**

**Test:**

Verifies that providing an incorrect password for an existing user fails authentication and returns False.

**Coverage:**

Ensures that the verification function properly distinguishes between correct and incorrect passwords. It covers the integrity of the hashing and salt mechanism and confirms that the system rejects invalid credentials.

**Test Verifying Nonexistent User (test\_verify\_nonexistent\_user):**

**Test:**

Attempts to verify a user who is not in the file, expecting the verification to return False.

**Coverage:**

Checks how the system handles lookups for users that were never added, ensuring it gracefully fails and does not produce errors. This validates handling of negative lookup scenarios.

**Test Unique Salts for Users with the Same Password (test\_unique\_salt\_for\_users\_with\_same\_password):**

**Test:**

Adds two different users with the same password and verifies that they have distinct salts.

**Coverage:**

Confirms that each user entry is independently protected, ensuring that even if identical passwords are used, the resulting hash values differ due to unique salts. This test addresses a critical security requirement (rainbow table protection) and ensures proper salt generation.

1. c.

**Test: test\_valid\_login\_and\_permissions**

**Test:**

Authenticates a valid user (‘SashaKim’) with the correct password.

Verifies the role (‘Client’) and assigned permissions.

Checks specific permissions related to the role.

**Coverage:**

Valid login scenario.

Rolebased permission assignment and access verification for a ‘Client’.

**Test name: test\_invalid\_login**

**Test:**

Attempts login with an invalid username.

Attempts login with an incorrect password for a valid username.

Verifies the error messages returned for both scenarios.

**Coverage:**

Handling of invalid login attempts:

Nonexistent username.

Incorrect password for an existing user.

**Test name: test\_access\_rights\_client**

**Test:**

Authenticates a ‘Client’ user (‘SashaKim’) and verifies rolespecific permissions:

Ensures access to ‘view\_account\_balance’.

Denies access to permissions like ‘modify\_investment\_portfolio’.

**Coverage:**

Permission validation for the ‘Client’ role.

**Test name: test\_access\_rights\_premium\_client**

**Test:**

Authenticates a ‘Premium Client’ user (‘NoorAbbasi’) and verifies:

Access to advanced permissions like ‘modify\_investment\_portfolio’.

Restricted access to unrelated permissions, such as ‘view\_private\_consumer\_instruments’.

**Coverage:**

Permission validation for the ‘Premium Client’ role.

**Test name: test\_access\_rights\_financial\_advisor**

**Test:**

Authenticates a ‘Financial Advisor’ user (‘MikaelChen’) and verifies:

Access to permissions like ‘view\_private\_consumer\_instruments’.

Denial of permissions outside the role

**Coverage:**

Permission validation for the ‘Financial Advisor’ role.

**Test name: test\_access\_rights\_financial\_planner**

**Test:**

Authenticates a ‘Financial Planner’ user (‘EllisNakamura’) and verifies:

Access to permissions like ‘view\_money\_market\_instruments’ and ‘modify\_any\_investment\_portfolio’.

**Coverage:**

Permission validation for the ‘Financial Planner’ role.

Multiple permission checks for a single role.

**Test name: test\_access\_rights\_teller\_business\_hours**

**Test:**

Authenticates a ‘Teller’ user (‘AlexHayes’) during business hours.

Verifies permissions such as ‘view\_any\_account\_balance’.

**Coverage:**

Permission validation for the ‘Teller’ role under business hours conditions.

**Test name: test\_teller\_outside\_business\_hours**

**Test:**

Overrides the ‘is\_business\_hour’ function to simulate outside business hours.

Authenticates a ‘Teller’ user (‘AlexHayes’) and verifies restricted permissions.

Restores the original ‘is\_business\_hour’ function after the test.

**Coverage:**

Conditional access control based on timesensitive permissions for the ‘Teller’ role.

Dynamic behavior for business hourspecific permissions.