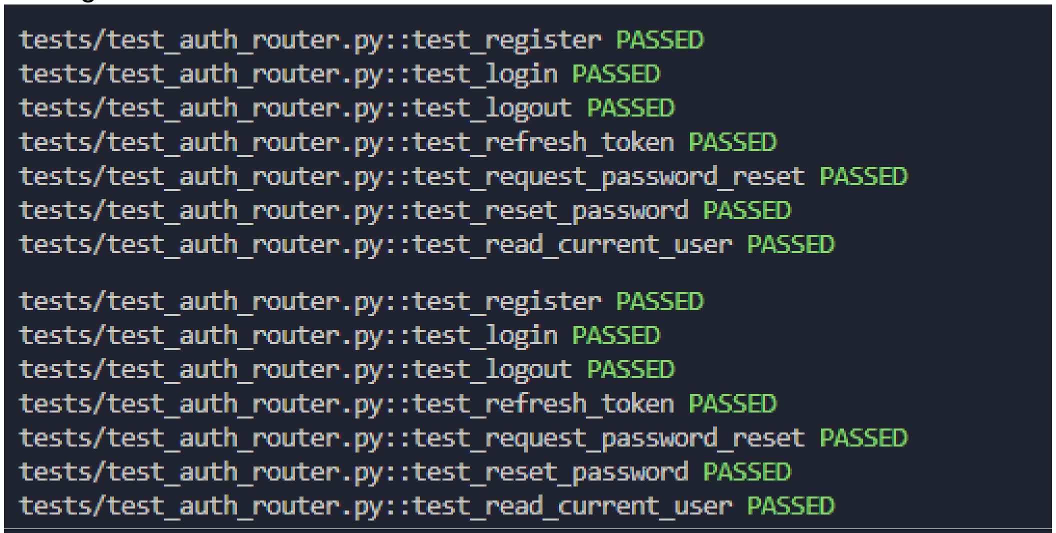
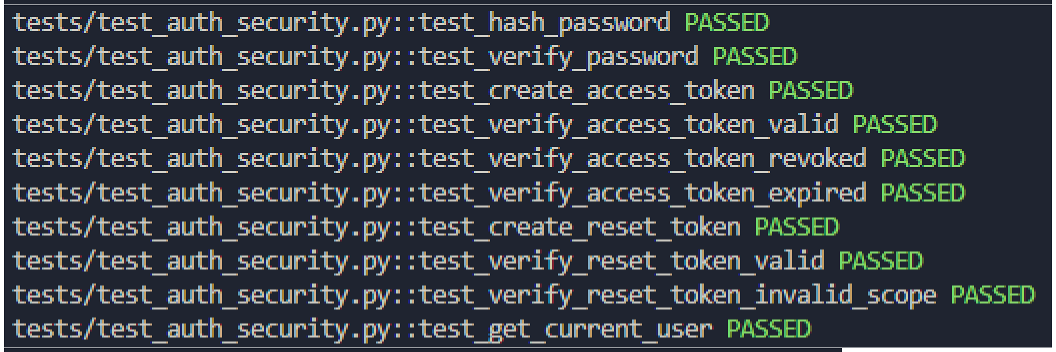
Testing

Testing Authorization

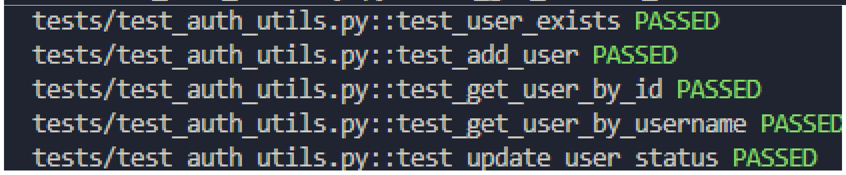


```
tests/test_auth_router.py::test_register PASSED
tests/test_auth_router.py::test_login PASSED
tests/test_auth_router.py::test_logout PASSED
tests/test_auth_router.py::test_refresh_token PASSED
tests/test_auth_router.py::test_request_password_reset PASSED
tests/test_auth_router.py::test_reset_password PASSED
tests/test_auth_router.py::test_read_current_user PASSED
```

```
tests/test_auth_router.py::test_register PASSED
tests/test_auth_router.py::test_login PASSED
tests/test_auth_router.py::test_logout PASSED
tests/test_auth_router.py::test_refresh_token PASSED
tests/test_auth_router.py::test_request_password_reset PASSED
tests/test_auth_router.py::test_reset_password PASSED
tests/test_auth_router.py::test_read_current_user PASSED
```



```
tests/test_auth_security.py::test_hash_password PASSED
tests/test_auth_security.py::test_verify_password PASSED
tests/test_auth_security.py::test_create_access_token PASSED
tests/test_auth_security.py::test_verify_access_token_valid PASSED
tests/test_auth_security.py::test_verify_access_token_revoked PASSED
tests/test_auth_security.py::test_verify_access_token_expired PASSED
tests/test_auth_security.py::test_create_reset_token PASSED
tests/test_auth_security.py::test_verify_reset_token_valid PASSED
tests/test_auth_security.py::test_verify_reset_token_invalid_scope PASSED
tests/test_auth_security.py::test_get_current_user PASSED
```



```
tests/test_auth_utils.py::test_user_exists PASSED
tests/test_auth_utils.py::test_add_user PASSED
tests/test_auth_utils.py::test_get_user_by_id PASSED
tests/test_auth_utils.py::test_get_user_by_username PASSED
tests/test_auth_utils.py::test_update_user_status PASSED
```

The following tests related to Authorization features use predominantly mocking by fake user information and forcing other methods to return predetermined values to test only one method at a time.

1. Test_regiseter
   Preconditions:
   a. User exists
   b. User is valid

   Steps:
   a. Create mock user
   b. Add valid mock user
   c. Register mock user

   Expected results:
   a. Mock user can be retrieved
   b. Mock user is registered correctly

   Status: PASSED
   Type: Mock, White-box, Unit Test
   as the user being registered is a predetermined user and outside methods that are not part of the test have a forced return value to match the test conditions and are not directly used. This allows the test to isolate the endpoint being tested from others

2. Test_login
   Preconditions:
   a. User exists
   b. User has password
   c. User has access token

   Steps:
   a. Get mock user from database
   b. Verify mock user password
   c. Give mock user access token
   d. Set mock user as logged in

   Expected results:
   a. Access token has valid token value
   b. Token type is bearer

   Status: PASSED
   Type: Mock, White-box, Unit test
   As the user being logged in is a predetermined user and outside methods that are not part of the test have a forced return value to match test conditions thus eliminating any outside dependencies

3. Test_logout
   Preconditions:
   a. User is already logged in

Steps:

a. Revoke user token

Expected results:

a. User token is set to an invalid token
b. Valid token is correctly replaced with invalid token

Status: PASSED

Type: Mock, white-box, Unit test

The revoke_token method return is mocked thus isolating the endpoint making it a unit test and since it knows the revoke_tokens method it is a white-box test

4. Test_refresh_token

Preconditions:

a. User has a token

Steps:

a. Create new token
b. Get current user
c. Replace current token with new token

Expected results:

a. A new token is correctly made
b. The token of the current user is currently replaced with the new token
c. Token is correctly saved with the user data

Status: PASSED

Type: Mock, white-box, unit test

The mock access token isolates the endpoint and the dependency on the current user token is overridden, since the create_access_token is known it is a white box test

5. Test_request_password_reset

Preconditions:

a. User exists

Steps:

a. Load all users
b. Create password reset token
c. Give token to current user

Expected results:

a. User gets a password reset token of the correct format that can be used later to reset their password

Status: PASSED

Type: Mock, white-box, unit test

Internal methods are used with mocked user data, the mocked endpoints isolate the endpoint being tested this being a unit test

6. Test_reset_password

Preconditions:

    a. User exists
    b. User have password reset token

  Steps:

    a. User gets valid token
    b. User gets new password hash
    c. User password is replaced with new password
    d. New user passwords gets correctly hashed and stored

  Expected results:

    a. User password is replaced with a new password
    b. New user password is saved correctly

  Status: PASSED

  Type: Mock, white-box. Unit test

  Uses several internal method values are mocked with user data isolating the endpoint thus being a white-box unit test

7. Test_read_current_user

  Preconditions:

    a. User is valid
    b. User is logged in

  Steps:

    a. Create mock current user
    b. Override current user with mock user
    c. Get current user details
    d. Check current mock user details

  Expected results:

    a. Current user details match the created mock user details

  Status: PASSED

  Type: Mocking, white-box, unit test

  Instead of patching it directly overwrites the the current user data with a mock User. This isolates it from other dependencies turning it in a white-box unit test as it does not rely on any other methods

8. Test_user_exists

  Preconditions:

    a. Some users already exist in the system.

  Steps:

    a. Mock load_all_users to return a list of existing users.
    b. Call user_exists with a username that already exists.
    c. Call user_exists with an email that already exists.
    d. Call user_exists with a new username and email.

  Expected results:

    a. Returns True and "Username already taken" if username exists.
    b. Returns True and "Email already taken" if email exists.
    c. Returns False and None if neither exists.

Status: PASSED
Type: Mock, white-box, unit test
Tests validation logic for username and email uniqueness using mocked data.

9. Test_add_user
Preconditions:
    a. No active users exist.
Steps:
    a. Mock load_active_users to return an empty list.
    b. Mock save_active_users to capture saved data.
    c. Call add_user with a new user and active=True.
Expected results:
    a. Active users list is updated with the new user.
    b. The new user includes a penalties field.
Status: PASSED
Type: Mock, white-box, unit test
Tests adding a new user to active users storage using mocked dependencies.

10. Test_get_user_by_id
Preconditions:
    a. Some users exist in the system.
Steps:
    a. Mock load_all_users to return a list of users.
    b. Call get_user_by_id with a valid user ID.
    c. Call get_user_by_id with a non-existent user ID.
Expected results:
    a. Returns the correct user for a valid ID
    b. Returns None for a non-existent ID.
Status: PASSED
Type: Mock, white-box, unit test
Tests retrieving users by ID using mocked storage.

11. Test_get_user_by_username
Preconditions:
    a. Some users exist in the system.
Steps:
    a. Mock load_all_users to return a list of users.
    b. Call get_user_by_username with an existing username.
    c. Call get_user_by_username with a username that does not exist.
Expected results:
    a. Returns the correct user for an existing username.
    b. Returns None if the username does not exist.
Status: PASSED
Type: Mock, white-box, unit test

Tests retrieving users by username using mocked storage.

12. Test_update_user_status
    Preconditions:
    a. Some users exist as active. Inactive users list is empty.
    Steps:
    a. Mock load_active_users and load_inactive_users with current lists.
    b. Mock save_active_users and save_inactive_users to capture saved data.
    c. Call update_user_status to set a user to inactive.
    Expected results:
    a. User is removed from active users list.
    b. User is added to inactive users list with updated status.
    Status: PASSED
    Type: Mock, white-box, unit test
    Tests status update logic and storage handling using mocked dependencies.

13. Test_hash_password
    Preconditions:
    Password hashing utility is available.
    Steps:
    1. Mock pwd_context.hash.
    2. Call hash_password with a sample password.
    Expected results:
    Returns the mocked hash value. Hash function is called once with the correct password.
    Status: PASSED
    Type: Mock, white-box, unit test
    Tests password hashing logic using a mocked hash function.

14. Test_verify_password
    Preconditions:
    Password verification utility is available.
    Steps:
    1. Mock pwd_context.verify.
    2. Call verify_password with a plain and hashed password.

    Expected results:
    Returns True. Verify function is called once with correct arguments.
    Status: PASSED
    Type: Mock, white-box, unit test
    Tests password verification logic using mocked dependencies.

15. Test_create_access_token
    Preconditions:

JWT creation utility is available.

Steps:

1. Mock jwt.encode.

2. Call create_access_token with a payload and expiration.
   Expected results:
   Returns the mocked token. JWT encode function is called.
   Status: PASSED
   Type: Mock, white-box, unit test
   Tests JWT token creation using mocked encoding.

16. Test_verify_access_token_valid
    Preconditions:
    Valid, non-revoked JWT token exists.
    Steps:

    1. Mock is_token_revoked to return False.

    2. Mock jwt.decode to return a valid payload.

    3. Call verify_access_token with the token.
       Expected results:
       Returns payload with correct sub.
       Status: PASSED
       Type: Mock, white-box, unit test
       Tests verification of a valid access token.

17. Test_verify_access_token_revoked
    Preconditions:
    JWT token is revoked.
    Steps:

    1. Mock is_token_revoked to return True.

    2. Call verify_access_token with the token.
       Expected results:
       Returns None since token is revoked.
       Status: PASSED
       Type: Mock, white-box, unit test
       Tests handling of revoked access tokens.

18. Test_verify_access_token_expired
    Preconditions:
    JWT token is expired.
    Steps:

    1. Mock is_token_revoked to return False.

    2. Mock jwt.decode to raise ExpiredSignatureError.

3. Call verify_access_token with the token.
   Expected results:
   Returns None since token is expired.
   Status: PASSED
   Type: Mock, white-box, unit test
   Tests handling of expired access tokens.

19. Test_create_reset_token
    Preconditions:
    JWT reset token creation utility is available.
    Steps:

    1. Mock jwt.encode.

    2. Call create_reset_token with a user ID.
       Expected results:
       Returns mocked reset token. JWT encode function is called.
       Status: PASSED
       Type: Mock, white-box, unit test
       Tests reset token creation logic.

20. Test_verify_reset_token_valid
    Preconditions:
    Valid password reset token exists.
    Steps:

    1. Mock jwt.decode to return a payload with correct scope.

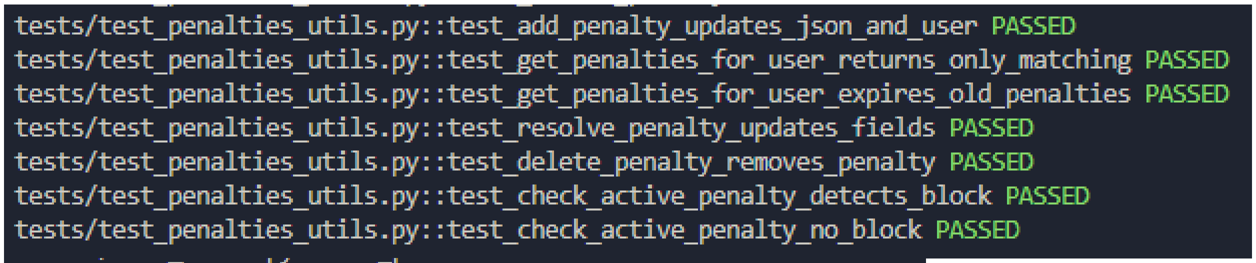
    2. Call verify_reset_token with the token.
       Expected results:
       Returns the user ID from the token.
       Status: PASSED
       Type: Mock, white-box, unit test
       Tests verification of valid reset tokens.

21. Test_verify_reset_token_invalid_scope
    Preconditions:
    Reset token exists but has wrong scope.
    Steps:

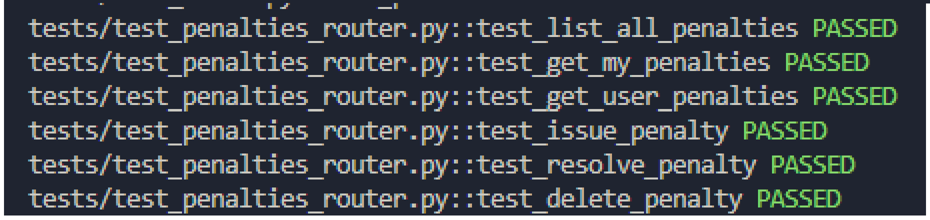
    1. Mock jwt.decode to return payload with incorrect scope.

    2. Call verify_reset_token with the token.
       Expected results:
       Returns None since scope is invalid.
       Status: PASSED
       Type: Mock, white-box, unit test
       Tests handling of invalid reset tokens.

Testing Penalties



```
tests/test_penalties_utils.py::test_add_penalty_updates_json_and_user PASSED
tests/test_penalties_utils.py::test_get_penalties_for_user_returns_only_matching PASSED
tests/test_penalties_utils.py::test_get_penalties_for_user_expires_old_penalties PASSED
tests/test_penalties_utils.py::test_resolve_penalty_updates_fields PASSED
tests/test_penalties_utils.py::test_delete_penalty_removes_penalty PASSED
tests/test_penalties_utils.py::test_check_active_penalty_detects_block PASSED
tests/test_penalties_utils.py::test_check_active_penalty_no_block PASSED
```



```
tests/test_penalties_router.py::test_list_all_penalties PASSED
tests/test_penalties_router.py::test_get_my_penalties PASSED
tests/test_penalties_router.py::test_get_user_penalties PASSED
tests/test_penalties_router.py::test_issue_penalty PASSED
tests/test_penalties_router.py::test_resolve_penalty PASSED
tests/test_penalties_router.py::test_delete_penalty PASSED
```

The following tests are related to verifying penalty features

The following tests related to Authorization features use predominantly mocking with fake penalty information and forcing other methods to return predetermined values to test only one method at a time.

1. Test_add_penalty_updates_json_and_user

   Preconditions:

   a. Have saved users

   Steps

   a. Load mock user
   b. Add mocked penalty to user
   c. Save penalty to correct user

   Expected results

   a. Penalty gets added to selected mocked user

   Status: PASSED

   Type: Mock, white-box, unit test

   Since the dependencies are mocked and the penalty added is a mock object the only method tested is add_penalty making it a unit test

2. Test_get_penalties_for_user_returns_only_matching

   Preconditions

   a. Have saved penalties in JSON

   Steps

   a. Find mock user that has penalty by id

   Expected results

   a. Returns the penalty of a specific user

   Status: PASSED

   Type: Mock, white-box, unit test

   Since only the get_penalties_for_users is tested it is a white-box unit test

3. Test_get_penalties_for_user_expires_old_penalties
   Preconditions
   a. Have saved penalties in JSON
   b. Have some penalties be expired

   Steps
   a. Mock expiration time
   b. Mock penalties with mock expiration time
   c. Find mock user that has expired penalties

   Expected results
   a. Returns the expired penalties of a specific user

   Status: PASSED
   Type: Mock, white - box, unit test
   Expiration handling logic uses mocked dependencies and only uses get_penalties_for_users
4. Test_resolve_penalty_updates_fields
   Preconditions:
   a. Have saved penalties in JSON

   Steps:
   a. Load mock penalties
   b. Find penalty of mock user
   c. Resolve penalty by mock moderator
   d. Save penalty as resolved in JSON

   Expected results
   a. Penalties linked to specific user are removed from JSON

   Status: PASSED
   Type: Mock, white - box, Integration test
   Uses mocking for penalties and user data and tests using internal method _unlink_penalty_from_user

5. Test_delete_penalty_removes_penalty
   Preconditions:
   a. Have saved penalties in JSON

   Steps
   a. Load penalties
   b. Find penalties by penalty id
   c. Delete all matching penalties

   Expected results
   a. Penalties matching all penalty id's are removed from JSON

   Status: PASSED
   Type: Mock, white - box, Integration test
   Uses mocking for penalties and user data and test method uses other internal method _unlink_penalty_from_user

6. Test_check_active_penalty_detects_block

   Preconditions
   a. Have penalties saved in JSON
   b. Have some penalties be active

   Steps
   a. Mock penalty
   b. Find active penalty for user

   Expected results
   a. Returns any active penalties for a user

   Status: PASSED

   Type: Mock, white - box, Integration Test

   Mocks user penalty, and method being tested uses other internal methods

7. Test_check_active_penalty_no_block

   Preconditions:
   a. Have penalties saved in JSON
   b. Have some penalties be no active

   Steps
   a. Mock penalty
   b. Find active penalty for user

   Expected results
   b. Returns any resolved or expired penalties for a user

   Status: PASSED

   Type: Mock, white-box, integration test

   Mocks user penalty, and method being tested uses other internal methods

8. Test_list_all_penalties

   Preconditions:
   a. Admin user is logged in.
   b. Penalties exist in the system.

   Steps:
   a. Override get_current_user to admin.
   b. Load fake penalties.
   c. Send GET /penalties/ request.

   Expected results:
   a. Returns status 200 and all penalties.

   Status: PASSED

   Type: Integration, white-box test

   Tests endpoint behavior with dependency overrides and mocked data.

9. Test_get_my_penalties

   Preconditions:
   a. Member user is logged in.

b. Member has penalties assigned.

Steps:

a. Override get_current_user to member.
b. Monkeypatch get_penalties_for_user to return mocked penalties.
c. Send GET /penalties/me request.

Expected results:

a. Returns status 200 and only penalties belonging to the member.

Status: PASSED

Type: Integration, white-box test

Tests endpoint returns only user-specific penalties.

10. Test_get_user_penalties

Preconditions:

a. The moderator user is logged in.
b. Penalties exist for a specific user.

Steps:

a. Override get_current_user to moderator.
b. Monkeypatch get_penalties_for_user to return mocked penalties.
c. Send GET /penalties/{user_id} request

Expected results:

a. Returns status 200 and penalties for the specified user.

Status: PASSED

Type: Integration, white-box test

Tests moderator can retrieve penalties for any user.

11. Test_issue_penalty

Preconditions:

a. Moderator user is logged in.
b. The target user exists.

Steps:

a. Override get_current_user to moderator.
b. Monkeypatch add_penalty to return the given penalty
c. Send POST /penalties/ request with payload.

Expected results:

a. Returns status 200 and the newly issued penalty.

Status: PASSED

Type: Integration, white-box test

Tests issuing penalties via the endpoint using mocked add function.

12. Test_resolve_penalty

Preconditions:

a. The moderator user is logged in.
b. Penalty exists and is unresolved.

Steps:

a. Override get_current_user to moderator.
b. Monkeypatch resolve_penalty to capture call arguments.
c. Send PATCH /penalties/{penalty_id}?notes={text} request.

Expected results:

a. Returns status 200 and calls resolve_penalty with correct penalty attributes

Statues: PASSED

Type: Integration, white-box test

Tests resolving penalties via endpoint and ensures correct data is passed.

13. Test_delete_penalty

Preconditions:

a. Admin user is logged in. Penalty exists.

Steps:

a. Override get_current_user to admin.
b. Monkeypatch delete_penalty to capture deleted penalty ID.
c. Send DELETE /penalties/{penalty_id} request.

Expected results:

a. Returns status 200 and calls delete_penalty with correct penalty ID.

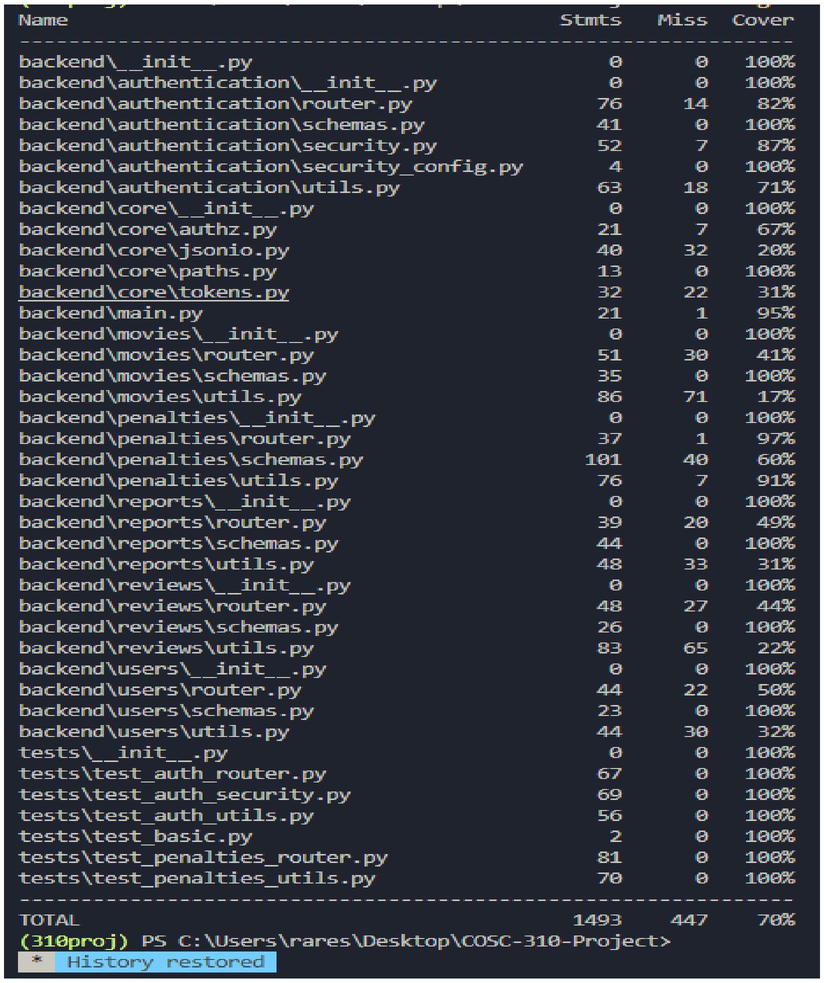
Status: PASSED

Type: Integration, white-box test

Tests admin can delete penalties through the endpoint.

Coverage

The Overall code coverage Is 54.8% excluding the 100% coverage from the tests. The various tests in authorization and penalties cover some of the other core components like Users. However, the other components need more stand alone testing to increase the percent coverage for each file.



```
Name                                            Stmts   Miss  Cover
-------------------------------------------------------------------
backend\__init__.py                                 0      0   100%
backend\authentication\__init__.py                  0      0   100%
backend\authentication\router.py                   76     14    82%
backend\authentication\schemas.py                  41      0   100%
backend\authentication\security.py                 52      7    87%
backend\authentication\security_config.py           4      0   100%
backend\authentication\utils.py                    63     18    71%
backend\core\__init__.py                            0      0   100%
backend\core\authz.py                              21      7    67%
backend\core\jsonio.py                             40     32    20%
backend\core\paths.py                              13      0   100%
backend\core\tokens.py                             32     22    31%
backend\main.py                                    21      1    95%
backend\movies\__init__.py                          0      0   100%
backend\movies\router.py                           51     30    41%
backend\movies\schemas.py                          35      0   100%
backend\movies\utils.py                            86     71    17%
backend\penalties\__init__.py                       0      0   100%
backend\penalties\router.py                        37      1    97%
backend\penalties\schemas.py                      101     40    60%
backend\penalties\utils.py                         76      7    91%
backend\reports\__init__.py                         0      0   100%
backend\reports\router.py                          39     20    49%
backend\reports\schemas.py                         44      0   100%
backend\reports\utils.py                           48     33    31%
backend\reviews\__init__.py                         0      0   100%
backend\reviews\router.py                          48     27    44%
backend\reviews\schemas.py                         26      0   100%
backend\reviews\utils.py                           83     65    22%
backend\users\__init__.py                           0      0   100%
backend\users\router.py                            44     22    50%
backend\users\schemas.py                           23      0   100%
backend\users\utils.py                             44     30    32%
tests\__init__.py                                   0      0   100%
tests\test_auth_router.py                          67      0   100%
tests\test_auth_security.py                        69      0   100%
tests\test_auth_utils.py                           56      0   100%
tests\test_basic.py                                 2      0   100%
tests\test_penalties_router.py                     81      0   100%
tests\test_penalties_utils.py                      70      0   100%
-------------------------------------------------------------------
TOTAL                                            1493    447    70%
(310proj) PS C:\Users\rares\Desktop\COSC-310-Project>
 *  History restored
```