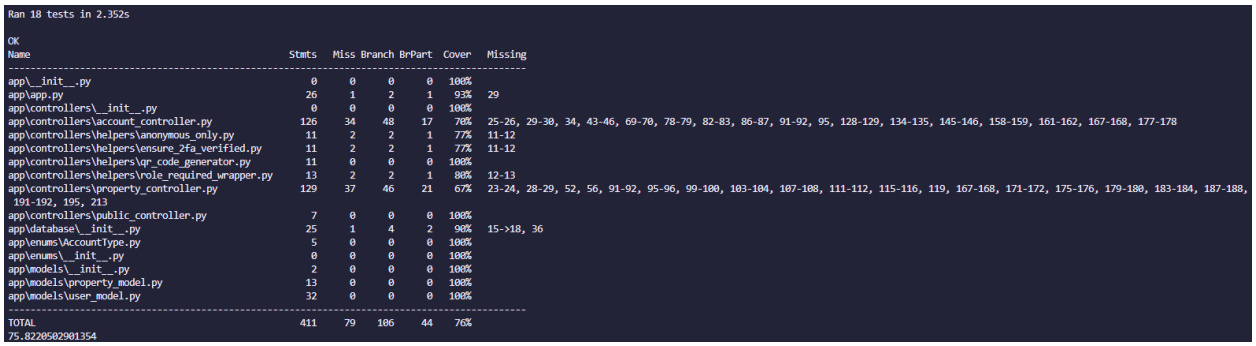
QUESTION:
Explain for each file which statement is missed in code coverage testing and why. Write your explanation in the same file as above

---

OLD COVERAGE ANALYSIS (ANALYSIS BELOW):



```
Ran 18 tests in 2.352s

OK
Name                                                Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------
app\__init__.py                                         0      0      0      0   100%
app\app.py                                             26      1      2      1    93%   29
app\controllers\__init__.py                             0      0      0      0   100%
app\controllers\account_controller.py                 126     34     48     17    70%   25-26, 29-30, 34, 43-46, 69-70, 78-79, 82-83, 86-87, 91-92, 95, 128-129, 134-135, 145-146, 158-159, 161-162, 167-168, 177-178
app\controllers\helpers\anonymous_only.py              11      2      2      1    77%   11-12
app\controllers\helpers\ensure_2fa_verified.py         11      2      2      1    77%   11-12
app\controllers\helpers\qr_code_generator.py           11      0      0      0   100%
app\controllers\helpers\role_required_wrapper.py       13      2      2      1    80%   12-13
app\controllers\property_controller.py                129     37     46     21    67%   23-24, 28-29, 52, 56, 91-92, 95-96, 99-100, 103-104, 107-108, 111-112, 115-116, 119, 167-168, 171-172, 175-176, 179-180, 183-184, 187-188,
 191-192, 195, 213
app\controllers\public_controller.py                    7      0      0      0   100%
app\database\__init__.py                               25      1      4      2    90%   15->18, 36
app\enums\AccountType.py                                5      0      0      0   100%
app\enums\__init__.py                                   0      0      0      0   100%
app\models\__init__.py                                  2      0      0      0   100%
app\models\property_model.py                           13      0      0      0   100%
app\models\user_model.py                               32      0      0      0   100%
-----------------------------------------------------------------------------------------------
TOTAL                                                 411     79    106     44    76%
75.8220502901354
```

---

app\app.py

LINE 29: `return None`

Here, we have 26 executables statements, where app.py has a total of 42 lines being used. Non-executables include 11 lines of blank code, 3 import lines and 2 decorators, bumping the number down to 26 executables as stated. From the analysis, we can see that line 29 is not executed. Looking at the block it's under, the `@login_manager.user_loader` function is responsible for loading users by their ID when Flask-Login processes a session. This function checks if `usr_id` provided is a digit, and if so, it retrieves and returns the corresponding `User` instance. If `usr_id` is not a valid digit or does not correspond to a valid user it returns `None`.

In the coverage analysis, the `return None` line shows as missing because no test scenario includes an invalid `usr_id` (like most cases for the following missing statements in the analysis). To achieve full coverage, the test suite should include cases where `usr_id` is either a non-digit string or an invalid ID, ensuring that `usr_loader` returns `None` in these cases.

---

app\controllers\account_controller.py

Here, based on the information given, we have 126 executables statements. The following are the missing executables that were not accounted for when testing:

These lines handle validation errors in the `Login` route. If specific invalid inputs (such as missing email/password) are not included, the error messages and flash calls won't execute.

LINE 25-26:

```
validation_error = True
flash("Please enter a valid email address to login.", 'error')
```

LINE 29-30:

```
validation_error = True
flash("Please enter a valid password to login.", 'error')
```

LINE 34:

```
return render_template('account/login.html', email=email)
```

Looking at `test_accounts.py`, if we go down to the method `test_invalid login`, we've tested with an invalid email and password, but the code does not test cases where either the email or password is completely missing, in other words, empty values. The missing lines need to handle these cases, displaying an error message if either field is left blank.

```
    def test_invalid_login(self):
        """Test Invalid login"""
        response = self.client.post('/login', data=dict(
            email='nonexistent@example.com',
            password='boguspassword'
        ), follow_redirects=True)
        self.assertIn(b'LOGIN', response.data)
        self.assertIn(b'The email or password provided is invalid!', response.data)
```

This validation ensures that a new registration email is unique.

LINE 43-46:

```
next_url = '/verify_2fa'
     if redirect_url:
          next_url += f'?next={redirect_url}'
     return redirect(next_url)
```

The missing coverage is due to a lack of test cases where the user is redirected to `/verify_2fa`, potentially with a `next` URL, which would trigger the execution of this redirection

These lines handle fields like names and password requirements in the `register` route.

LINE 69-70:

```
validation_error = True
            flash("Please enter a valid email address.", 'error')
```

LINE 78-79:

```
  validation_error = True
            flash('Please enter a first name.', 'error')
```

LINE 82-83:

```
 validation_error = True
            flash('Please enter a last name.', 'error')
```

In `test_accounts.py`, the `test_register_fail` method checks for names and an existing email, but there are no tests verifying empty or improperly formatted emails and names. Additionally, no tests attempt registration where the password and confirmation do not match.

LINE 86-87:

```
            flash('The password does not meet the requirements.', 'error')
            return render_template('account/register.html', first_name=first_name,
last_name=last_name,
                                   email=email, account_type=account_type)
```

LINE 91-92:

```
              validation_error = True
            flash('The confirmed password does not match.', 'error')
```

LINE 95:

```
            return render_template('account/register.html', first_name=first_name,
last_name=last_name, email=email, account_type=account_type)
```

Despite having `test_register_success` and `test_register_fail`, these lines remain missing in the coverage because the tests do not include cases where the password fails to meet the criteria or passwords don't match. These validation checks are only triggered under specific failure conditions not present in `test_accounts.py`

These lines involve the `setup_2fa` route, particularly in cases where a user has already enabled 2FA or where the provided OTP is invalid.

LINE 128-129:

```
flash('You have already enabled 2FA Authentication and cannot access this page.',
category='error')
        return redirect('/')
```

The reason why these lines are missing in the test coverage is because `test_accounts.py` does not include a scenario where a user, with two factor authentication already enabled, attempts to access the `/setup_2fa` page again.

LINE 134-135:

```
flash("Please enter a valid OTP to completed 2FA verification.", category='error')
            return redirect(request.url)
```

The `test_setup_2fa` method checks valid OTP input but lacks tests for an incorrect OTP. The missing lines should handle invalid OTP cases, ensuring an error message appears in cases where the user inputs an invalid code.

These lines handle valid and redirection in `setup_2fa`, especially for cases with invalid OTPs or already verified sessions.

LINE 145-146:

```
flash('The OTP entered did not match the expected value. Please try again.',
category='error')
            return redirect(request.url)
```

In test_accounts.py, there isn't a case where an invalid OTP is entered during two factor authentication verification, which should trigger these lines. All tests in `test_setup_2fa` and `test_verify_2fa` assume a correct OTP is provided. To address this, there should be test cases that simulate a user entering an incorrect OTP.

LINE 158-159:

```
flash('You have not enabled 2FA Authentication. Enable it on this page first.',
category='error')
        return redirect('/setup_2fa')
```

LINE 161-162:

```
flash('You have already completed 2FA Authentication and cannot access this page.',
category='error')
        return redirect('/')
```

The above lines do not have test covering for cases where certain scenarios are not directly tested in `test_accounts.py`.

These lines handle redirecting after successful 2FA verification.

LINE 167-168:

```
flash("Please enter a valid OTP to completed 2FA verification.", category='error')
            return redirect(request.url)
```

LINE 177-178:

```
flash('The OTP entered did not match the expected value. Please try again.',
category='error')
            return redirect(request.url)
```

The above lines are missing because the test cases in `test_accounts.py` do not simulate the submission of an invalid/improperly formatted OTP, nor do they include scenarios where an incorrect but validly formatted OTP is submitted.

---

app\controllers\helpers\anonymous_only.py

LINE 11-12:

```
flash('The requested page is only accessible to logged out users.', 'error')
            return redirect('/')  # Forbidden
```

There are no tests in `test_accounts.py` where it encounters a scenario where the user is logged out, more specifically, there are no tests that attempt a route that uses `anonymous_only` and verify that the user is redirected with the flash message.

app\controllers\helpers\ensure_2fa_verified.py

LINE 11-12:

```
  flash('2FA Verification is required before accessing this page.', 'error')
            return redirect(f'/verify_2fa?next={request.path}')  # Forbidden
```

Looking through test_accounts.py, we have methods that relate the two factor authentication such as `test_setup_2fa` and `test_verify_2fa`, but none of them explicitly test `ensure_2fa_verified` file. From the methods, it is ensured that there is verification, but there aren't any direct tests that access a route protected by the file before the user verifies two factor authentication.

---

app\controllers\helpers\role_required_wrapper.py

LINE 12-13:

```
flash('The requested page requires different permissions to be accessed.', 'error')
return redirect('/')  # Forbidden
```

While `test_accounts.py` has several tests for different registration and log in, there are no tests that cover the case where a user attempts to access a page that required a specific role, specifically no tests that log in a user with a specific role nor check the access to a route protected by the `method roles_required` for any mismatch in roles.

---

app\controllers\property_controller.py

LINES 23-24:

```
            flash('No property IDs provided for deletion.', 'error')
            return redirect(url_for('property.get_properties'))
```

Based on `test_property_management.py`, tests are simulating a valid POST request where property IDs are always provided for deletion. As a result, the code block handling the case of missing property IDs is not triggered.

LINES 28-29:

```
            flash('Invalid property IDs provided.', 'error')
            return redirect(url_for('property.get_properties'))
```

In , the property IDs provided are always valid and numeric and since invalid property IDs are not being tested, this block of code does not execute.

LINE 52:

```
        return abort(404)
```

In `test_property_management.py`, the current tests are using valid property IDs that always return a found property, preventing any routes from this line being executed.

LINE 56:

```
        return abort(403)
```

In `test_property_management.py`, there are no tests that check whether the user owns the property or not, more specifically the testing file is currently set up in a manner where it only uses properties that belong to the user making the request, thus this condition is never triggered.

The following lines are missing in the coverage due to a lack of tests that do not check whether the submitted information is valid, invalid or missing. In other words, all user inputs will always be considered valid no matter the case.

LINES 91-92:

```
            flash('Please enter an address.', 'error')
            validation_error = True
```

LINES 95-96:

```
            flash('Please select a property type.', 'error')
            validation_error = True
```

LINES 99-100:

```
            flash('Please enter a valid square footage.', 'error')
            validation_error = True
```

LINES 103-104:

```
            flash('Please enter a valid number of bedrooms.', 'error')
            validation_error = True
```

LINES 107-108:

```
            flash('Please enter a valid number of bathrooms.', 'error')
            validation_error = True
```

LINES 111-112:

```
            flash('Please enter a valid rent price.', 'error')
            validation_error = True
```

LINES 115-116:

```
            flash('Please select availability status.', 'error')
            validation_error = True
```

LINES 167-168:

```
            flash('Please enter an address.', 'error')
            validation_error = True
```

LINES 171-172:

```
            flash('Please specify the property type.', 'error')
            validation_error = True
```

LINES 175-176:

```
            flash('Please enter a valid square footage.', 'error')
            validation_error = True
```

LINES 179-180:

```
            flash('Please enter a valid number of bedrooms.', 'error')
            validation_error = True
```

LINES 183-184:

```
            flash('Please enter a valid number of bathrooms.', 'error')
            validation_error = True
```

LINES 187-188:

```
            flash('Please enter a valid rent price.', 'error')
            validation_error = True
```

LINES 191-192:

```
            flash('Please select availability status.', 'error')
            validation_error = True
```

LINE 119:

```
            return render_template('managementProperties/property.html',
form_data=form_data, property=found_property)
```

LINES 195:

```
            return render_template('managementProperties/property.html',
form_data=form_data, property=None)
```

LINES 213:

```
    return render_template('managementProperties/property.html',
form_data=form_data, property=None)
```

Since the tests have not set up whether the data is valid, invalid or missing, it has yet to matter whether the condition is met where the submission fails due to validation errors.

---

app\database__init__.py

LINES 15-18:

```
if db_name_prefix != None:
        db_file_name = db_name_prefix + db_file_name


    db.db_path = os.path.join(db_directory, db_file_name)
```

The above lines are meant to configure where the database file will be stored, adding a unique prefix for test runs. Looking at `base_test_class.py` and other test files, coverage for these lines are missing because tests don't focus on this setup. but instead checks if the app functions correctly with the database.

LINE 36:

```
        upgrade()
```

The only line in the database `__init__.py` that doesn't get called is the `upgrade()` statement that migrates any old version of the database to the newest schema, since the unit test creates a new database at the beginning it always has the latest schema and never calls upgrade

NEW COVERAGE ANALYSIS:

```
Ran 39 tests in 4.600s

OK
Name                                                Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------
app\__init__.py                                         0      0      0      0   100%
app\app.py                                             26      0      2      0   100%
app\controllers\__init__.py                             0      0      0      0   100%
app\controllers\account_controller.py                 126     10     48      6    91%   45, 90-91, 94, 133-134, 160-161, 166-167
app\controllers\helpers\anonymous_only.py              11      0      2      0   100%
app\controllers\helpers\ensure_2fa_verified.py         11      0      2      0   100%
app\controllers\helpers\qr_code_generator.py           11      0      0      0   100%
app\controllers\helpers\role_required_wrapper.py       13      0      2      0   100%
app\controllers\property_controller.py                129      0     46      0   100%
app\controllers\public_controller.py                    7      0      0      0   100%
app\database\__init__.py                               24      1      4      2    89%   17->20, 37
app\enums\AccountType.py                                5      0      0      0   100%
app\enums\__init__.py                                   0      0      0      0   100%
app\models\__init__.py                                  2      0      0      0   100%
app\models\property_model.py                           13      0      0      0   100%
app\models\user_model.py                               32      0      0      0   100%
-----------------------------------------------------------------------------------------------
TOTAL                                                 410     11    106      8    96%
96.31782945736434
```

LIST OF NEW TEST CASES ADDED:

1. test_role_required_redirects_when_not_authenticated
2. test_role_required_redirects_when_wrong_role
3. Test_login_not_visible_logged_in
4. test_register_missing_info
5. Test_invalid_setup_2fa
6. test_setup_2fa_after_enabled
7. Test_invalid_verify_2fa
8. test_skip_2fa_verification
9. Test_verify_2fa_without_enabled
10. test_login_missing_info
11. Test_add_property_empty_fields
12. test_add_property_invalid_data
13. Test_view_property_missing_details
14. test_delete_properties_no_ids
15. Test_delete_properties_invalid_ids
16. test_view_property_other_user
17. Test_edit_property_invalid_data
18. test_add_property_invalid_availability
19. test_add_property_get
20. test_custom_not_found -> test_costum_not_found_1
21. test_custom_not_found_2
22. test_user_loader_with_invalid_id