CMPE 327 – Assignment 04

Department of Electrical and Computer Engineering
Queen's University

Composed By
Nicholas Seegobin (20246787)
Samhith Sripada (20232740)
Rodrigo Del Aguila (20275528)

Date of Submission
Tuesday, November 12th, 2024

"We hereby attest that the work submitted is our own individual work and that no portion of this submission has been copied in whole or in part from another source, with the possible exception of properly referenced material."

Table of Contents

Table of Figures

Step 01: Code Coverage Report

Login and Registration Tests - Coverage Analysis

The first report, seen in Figure 1, demonstrates the initial setup where no tests were executed, and the coverage report indicated "no data was collected." This can be attributed to the absence of critical setup files like a .coveragerc configuration and an initializing file for the test directory. Without these files, the coverage tool could not identify source files or tests for analysis.

```
---------- coverage: platform darwin, python 3.12.0-final-0 ----------
Name                                      Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------
apps/backend/models/userModel.py              8      8     0%   1-14
apps/backend/registration/register.py        47     47     0%   1-99
-----------------------------------------------------------------------
TOTAL                                        55     55     0%
```

Figure 1: A code coverage test for the userModel.py file and the register.py file showing 0% coverage.

The second report, seen in Figure 2, reflects some progress, as the test environment begins to detect test files like new_user.py and login.py. However, the test execution fails due to the error: "attempted relative import beyond top-level package."

This issue occurs because relative imports (e.g., from ..models.userModel import User) were used in the test files. Relative imports depend on the directory structure and how the script is executed. When pytest executes the tests, it treats the test files as standalone scripts, causing the relative imports to break. This issue indicates that the project structure and import strategy were not properly aligned with the testing framework.

The solution lies in transitioning from relative imports to absolute imports. Absolute imports use the full package path (e.g., from apps.backend.models.userModel import User), making them independent of the script's execution context. This change ensures that imports are resolved consistently, regardless of how the tests are executed.

Despite these issues, the introduction of the .coveragerc file in this stage allowed the coverage tool to start analyzing files like userModel.py and register.py. However, since the imports failed, the tests could not run, and coverage analysis was limited to static file detection without executing test cases.

```
---------- coverage: platform win32, python 3.12.6-final-0 -----------
Name                                                                                         Stmts   Miss  Cover   Missing
--------------------------------------------------------------------------------------------------------------------------
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\__init__.py                    0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\__init__.py            0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\models\__init__.py     0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\models\userModel.py    8      5    38%   6-9, 14
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\registration\__init__.py     0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\registration\register.py   47     44     6%   7-99
--------------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                           55     49    11%
```

Figure 2: A code coverage test for the userModel.py file and the register.py file showing 11% coverage.

The third report, seen in Figure 3, marks a turning point in the testing process. By resolving the import issues through the adoption of absolute imports, the tests successfully execute, and meaningful coverage results are generated. Absolute imports (e.g., from apps.backend.models.userModel import

User) ensure that the test files can locate the required modules consistently within the project hierarchy, regardless of the execution context.

The coverage summary shows that the tests provide 88% coverage for userModel.py and 93% coverage for register.py, indicating that most code paths are now tested. Additionally, the coverage analysis shows the specific lines of uncovered code, which provides actionable insights for achieving full coverage.

```
---------- coverage: platform win32, python 3.12.6-final-0 -----------
Name                                                                                      Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------------------------------------------
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\__init__.py                0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\__init__.py        0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\models\__init__.py 0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\models\userModel.py     8      1    88%   14
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\registration\__init__.py   0      0   100%
C:\Users\nicholas\Desktop\Cisc327-F24-Group17-SI\Assignment-4\apps\backend\registration\register.py  45      3    93%   60-62
-----------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                        53      4    92%
```

Figure 3: A code coverage test for the userModel.py file and the register.py file showing 93% coverage.

Tenant Tests - Coverage Analysis

The tenant tests, executed using a JavaScript testing framework, focus on validating the frontend functionality. There are 15 passing tests covering key areas like dropdown interactions, modal handling, table sorting, and event propagation. These tests achieve 100% coverage, including all statements, branches, functions, and lines in the Tenants.js file, ensuring thorough validation of user interface components. The coverage results can be seen in Figure 4.

The combination of C8, Mocha, and JSDOM was essential for these tests. C8 provided detailed and accurate coverage reports, highlighting untested paths and helping improve test quality. Mocha offered a clear and organized structure with describe and it blocks, making tests easy to write and maintain. JSDOM simulated the DOM, allowing thorough testing of UI interactions like dropdowns, modals, and table sorting. Sinon's fake timers made it easy to test time-based features, such as toast messages.

```
> tenant-tests@1.0.0 coverage
> nyc --reporter=text-summary --reporter=html mocha



  Tenants.js functionality
    Dropdown functionality
      ✔ should toggle dropdown menu on button click
      ✔ should close all other dropdowns when opening a new one
      ✔ should close dropdowns when clicking outside
    Modal functionality
      View Profile Modal
        ✔ should open the view-profile modal when button is clicked
        ✔ should close the view-profile modal when X is clicked
      Send Message Modal
        ✔ should open the send-message modal when button is clicked
        ✔ should close the send-message modal when Cancel is clicked
        ✔ should show toast message when Send is clicked and hide after 3 seconds
      Message History Modal
        ✔ should open the message-history modal when button is clicked
        ✔ should close the message-history modal when X is clicked
    Table sorting functionality
      parseDate function
        ✔ should correctly parse date strings
        ✔ should handle different months correctly
      Table sorting
        ✔ should sort text columns ascending and descending
        ✔ should sort dates correctly
    Event propagation
      ✔ should stop propagation on dropdown toggle click


  15 passing (302ms)


=============================== Coverage summary ===============================
Statements   : 100% ( 8/8 )
Branches     : 100% ( 0/0 )
Functions    : 100% ( 0/0 )
Lines        : 100% ( 8/8 )
================================================================================
```

Figure 4: A code coverage test for the tenants.js file showing 100% coverage.

Step 02: Missed Statements in Code Coverage

userModel.py

Missed Statement: Line 14 (return {...} in the to_dict method). See Figure 5 for more context.

Reason for Missing Coverage: The to_dict method is not invoked in any of the test cases. While the tests validate user attributes like username, email, password, and user_type, they do not explicitly check if the object can be successfully converted to a dictionary representation.

Solution: Add a unit test that creates a User object, calls the to_dict method, and validates the returned dictionary against the expected values. This will ensure the method's functionality is tested.

```
10
11      def to_dict(self):
12
13          # convert the object into a dictionary to store into database
14          return {
15              "username": self.username,
16              "email": self.email,
17              "password": self.password,
18              "user_type": self.user_type,
19          }
20
```

Figure 5: A snippet of code from userModel.py that shows the missed statement on line 14.

register.py

Missed Statements: Lines 60-62 (exception handling block in setUserData). See Figure 6 for more context.

Reason for Missing Coverage: These lines are part of the error-handling logic when attempting to insert a duplicate user into the database. Currently, no test case triggers an sqlite3.IntegrityError, so this block is never executed.

Solution: Add a test case that attempts to insert a User object with duplicate username or email into the database. This will cause the IntegrityError to be raised, ensuring this block is covered.

```
50
51      try:
52          cursor.execute(
53              f"""
54              INSERT INTO {table} (username, email, password, user_type)
55              VALUES (?, ?, ?, ?)
56          """,
57              (newUser.username, newUser.email, newUser.password, newUser.user_type),
58          )
59          conn.commit()
60      except sqlite3.IntegrityError:
61          print("ERROR: There was an issue creating an account")
62          return False  # Username or email already exists
63      finally:
64          # print the new user details to show the new object.
65          print("New User Data Set")
66          conn.close()
67      return True
68
69
```

Figure 6: A snippet of code from register.py that shows the missed statement on lines 60 - 62.