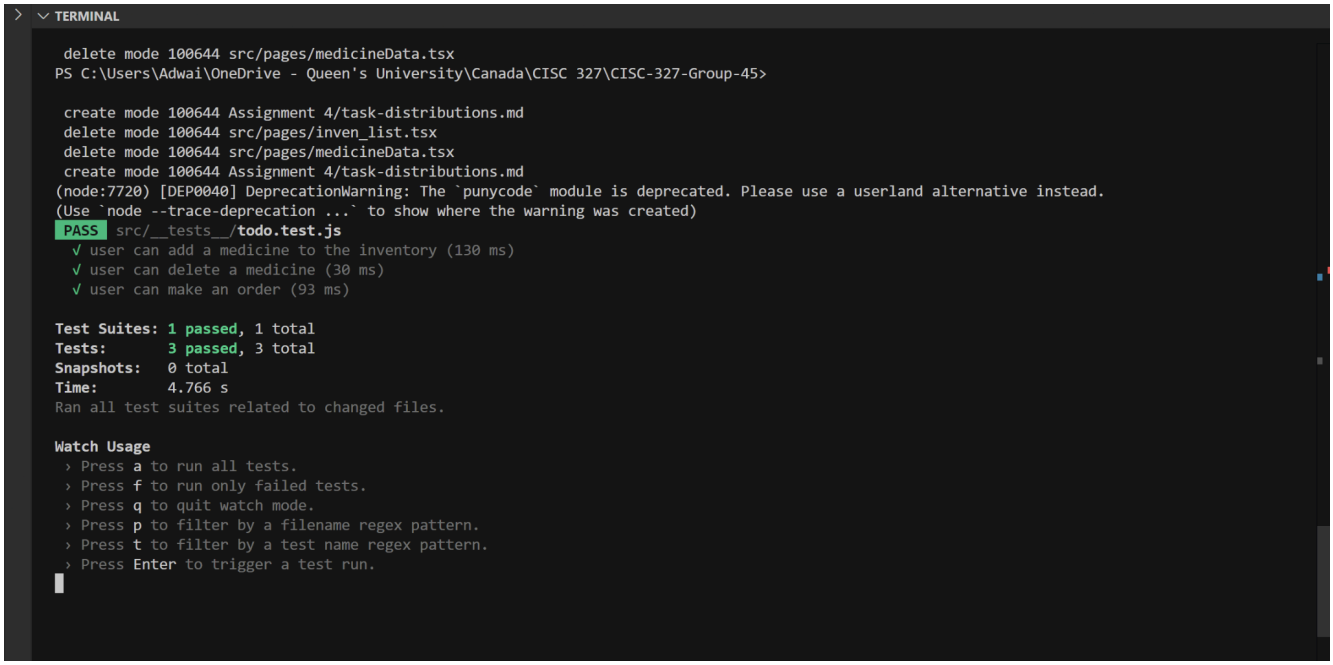
CISC 327
Assignment 5

In our previous testing, we conducted not only unit tests but also included integration tests, with 10 out of the 20 test cases being integration tests. From these, we have selected 3 test cases to showcase, which were already implemented in the earlier test runs. The attached screenshots correspond to these 3 specific test cases. To enhance clarity and focus, we will comment out the remaining 17 test cases, ensuring easier visibility of the selected tests.

Screenshot of the test scripts output:



Integration Tests

1. User Can Add a Medicine to the Inventory

- **Purpose:** This test ensures that users can successfully add a new medicine to the pharmacy inventory.
- **Flow:**
    1. The test first sets up a mock Firebase database (using getDoc) to simulate the existing inventory of medicines.
    2. The AddMedicine component is rendered, which represents the UI form where users enter details to add a new medicine.

3. We then simulate the below inputs in the form:
    i. Entering "Aspirin" in the medicine name field.
    ii. Enter "10" in the quantity field.
    iii. Enter a unique ID "A69U3490ID3493".
4. The "Add Medicine" button is clicked to test form submission.
5. After adding the medicine, the Inventory component is rendered to check if the new medicine appears in the list.
6. Finally, we verify that if "Aspirin" is displayed in the inventory, which confirms that we have successfully integrated the Add Medicine and Inventory.

This Integration test confirms that the add-medicine feature works end-to-end from accepting user input to updating the Inventory page.

2. User Can Delete a Medicine

- **Purpose:** This test ensures that there is integration between removing a medicine and updating the changes to the inventory.
- **Flow:**
    1. We set up a test by setting up a mock Firebase database with an initial list of dummy medicine data.
    2. The Inventory component is rendered, showing the current list of medicines.
    3. The test simulates a user clicking the delete button next to "Paracetamol."
    4. After the delete action, the test checks the Inventory UI again.
    5. It verifies that "Paracetamol" is no longer displayed, confirming that the medicine was successfully deleted.

This ensures that deletion properly updates the Firebase and reflects correctly in the frontend UI.

3. User Can Make an Order

- **Purpose:** This test ensures that users can create a new order and that it’s integrated with the order details in the sales report.
- **Flow:**
    1. The test sets up a mock Firebase database with available medicines and order details.
    2. The Order component is rendered, which allows users to create a new order.

3. We then simulate the below inputs in the order form:
    i. Enter "Aspirin" in the medicine name field.
    ii. Enter "32" in the quantity field.
    iii. Enter "H3DSFKP340Y9JS" in the order ID field.
    iv. Enter "79" as the price per unit.
4. After submitting the order form, the Sales component is rendered to check the updated sales report.
5. The test verifies that the new order appears in the sales report:
    i. Checks that "Aspirin," its ID, quantity, and price are displayed in the UI.

This test validates the integration between the Order component (user input) and the Sales component (displaying orders).

Instruction on How to Run the Tests:
To execute the test scripts, which are written with react-testing-library and Jest, launch a terminal server. After you have installed the dependencies, run:

```
npm run test
```

And then select the option for running all tests. This will run all the tests in the project.

Task Distribution Table:

Team Member	Contribution
Adwait	Formatting Test Scripts and Document Preparation
Ahmed	Setting up Jest Test Coverage and Document Preparation
James	Making the Integration Test Scripts