COMP 4004 A/B – Fall 2025

Assignment 3 — Web Acceptance Testing with Cypress

Due: December 5, 2025, at 23:59 on Brightspace

Overview

Assignment 3 builds directly on Assignment 2, using the same GitHub repository. This assignment extends your library management system by:

1. **Converting your Java business logic** (NOT your tests) into a Node.js/Express RESTful web application
2. **Creating a web-based user interface** using HTML/JavaScript that interacts with your REST API
3. **Writing Cypress end-to-end tests** that verify 3 acceptance scenarios through the UI layer

There is no need to rewrite your business logic; instead, wrap it with a web interface and test through that interface.

Learning Objectives

1. Learn to use Cypress for end-to-end UI testing.
2. Understand the differences between testing business logic directly (A2) versus testing through the UI layer (A3)
3. Practice creating maintainable UI tests using Cypress best practices
4. Experience converting a Java application to a Node.js web application

---

What You're Converting **(IMPORTANT - Read Carefully)**

DO Convert from Assignment 2**:**

- Your Java business logic classes (Book, User, Library, BorrowingService, HoldQueue, etc.)
- Your core functionality (borrow, return, place holds, notifications, borrowing limits)
- Your business rules and validation logic

DO NOT Convert from Assignment 2:

- Your Cucumber feature files (these stay in Java/A2)
- Your JUnit tests (these stay in Java/A1/A2)
- Your Cucumber step definitions (these stay in Java/A2)

What You're Building NEW for Assignment 3:

- Express.js REST API endpoints that expose your business logic
- HTML/JavaScript web frontend that calls these REST endpoints
- Cypress UI tests that interact with the web frontend (completely new tests)

Important Testing Clarification

Key Difference from A2

In Assignment 2, your Cucumber tests called business methods directly `(e.g., borrowBook(), returnBook()).`

In Assignment 3, your Cypress tests interact with the UI layer:

- Click buttons, fill forms, and navigate pages
- Verify visual feedback, messages, and page state
- Test the complete user experience from browser to server to business logic and back

What This Means

When the scenario describes user actions, your tests must interact through the UI. For example:

- **"user logs in"** → Your test fills in username/password fields and clicks "Login"
- **"selects Return Book"** → Your test clicks a "Return Book" button in the UI
- **"sees a message"** → Your test verifies text appears on the page
- **"notified"** → Your test checks for a notification banner, alert, or message displayed in the UI.

Your Cypress tests should (not an exhaustive list, and you don't necessarily need to do all of these):

- **Interact with DOM elements** (buttons, forms, lists, tables)
- **Verify UI state** (visible text, element presence, CSS classes)
- **Navigate between pages** if your app has multiple views

Before You Begin

You will need to convert your Java application from Assignment 2 into a web application. You may use AI tools to help convert your Java backend logic to a Node.js/Express.js application with a RESTful API. If you use an AI tool for any part of the conversion, you must indicate which AI tool(s) you used in your README file.

Your web application must run in **Chrome** and be functional; otherwise, **it will receive a grade of 0.** Your Web UI can be very basic, do not use any frameworks, because it becomes more challenging to use Cypress to test the UI elements

---

Prerequisites

1. **Complete A2 (if you have not completed it)**: Your library system business logic must work correctly for all core use cases
2. **Set up your development environment**:
   - Node 22.x LTS (JavaScript comes with the node installation)
   - npm (comes with Node.js)
   - Express (NPM install for Node) 5.1.X
   - Cypress (NPM install for Node) 15.X

Initial Setup Steps

1. **Create a new branch**:
   `A3-Cypress-<LastName>`
   A branching strategy suggested by students in class. Create a new branch in your repository, convert your A2 assignment into a web app, and check in that code so your A3 stream does not use Java, JUnit, or the Cucumber framework. **It is up to you how you branch, but it must work on the TA's machine.**

2. **Initialize Node.js project** (if not already done):
   npm init -y

3. Install required dependencies (only these two dependencies):
   ```
   npm install express
   npm install cypress --save-dev
   ```

   Note: Heavy deductions if other libraries are used.

What You Must Do

1. Learn Cypress (Before Starting Implementation)

- Official Cypress documentation: https://docs.cypress.io/
- Review the lecture on Cypress UI Testing and review the SimpleDemo code on Brightspace.

Key Cypress concepts to understand (these concepts will be covered in the lecture):

- `cy.visit()`, `cy.get()`, `cy.contains()`
- Assertions: `.should()`, `.and()`

2. Build a Web Frontend

Create a simple but functional web interface for your library system using Node.js and Express.js. Your web UI should provide the same core functionality as the Java application from **Assignment 1** (user login, browsing items, checkout/return, etc.).

Requirements:

Your web app must support:

Authentication/Session Management

- Login page with username/password fields
- Logout functionality
- To implement session management, refer to the SimpleDemo code on BrightSpace. The demo code uses a **`currentUser`** variable on the server. The application is meant to be a single-user web application, where only one person can log in at a time (simplified for this assignment). **Do not import other npm session libraries.**

Core Library Operations (accessible through UI – replicating assignment 1 console UI requirements)

- **View Available Books**: Display all books with their availability status
- **Borrow a Book**: Button or form to borrow an available book
- **Return a Book**: Button or form to return a borrowed book
- **View My Borrowed Books**: Show the current user's borrowed books with due dates
- **Place Hold**: Allow users to place holds on unavailable books
- **View Notifications**: Show notifications for available reserved books
- You may have more or fewer operations listed here.

Visual Feedback Requirements (not an exhaustive list, just examples)

- Success messages (e.g., "Book borrowed successfully")
- Error messages (e.g., "Borrowing limit reached")
- Book availability indicators (Available/Checked Out/Reserved)
- User's current borrowed book count
- Due dates displayed in human-readable format

Important Notes:

- Your UI does **not** need to be graphically sophisticated (basic HTML forms and tables are fine).
- Focus on **functionality and testability**, not appearance
- Use HTML and element attributes to make it easier for Cypress to select.

3. Design Test Data

Review each acceptance test scenario and plan your test data.

Use the same baseline data from A2:

Users:

Username	Password
alice	pass123
bob	pass456
charlie	pass789

Books (20 total): The Great Gatsby, To Kill a Mockingbird, 1984, Pride and Prejudice, The Hobbit, Harry Potter, The Catcher in the Rye, Animal Farm, Lord of the Flies, Jane Eyre, Wuthering Heights, Moby Dick, The Odyssey, Hamlet, War and Peace, The Divine Comedy, Crime and Punishment, Don Quixote, The Iliad, Ulysses.

4. Implement Cypress Tests

Create **a** Cypress test file for 3 of the A2 scenarios (described in the section "Acceptance Tests to Develop"), adapted for UI testing.

File Structure:

Refer to the file structure in the lecture slides

Test Organization:

- All three scenarios must be in one spec file
- Use the structure shown below
- You may add beforeEach() and/or afterEach() hooks within this structure if needed for setup or cleanup

Required test structure (provide your own description):

```
describe('Library Book Management', () => {
  it('should describe what scenario 1 tests', () => {
      //Test scenario1 1
  })
  it('should describe what scenario 2 tests', () => {
     // Test scenario 2
  })
  it('should describe what scenario 3 tests', () => {
     // Test scenario 3
  })
})
```

Note: Replace the describe() and it() descriptions with meaningful descriptions based on your actual scenarios.

Acceptance Tests to Develop

******Note**: You must implement a reset endpoint to restore the web server environment to its initial state. Refer to the server.js /api/reset endpoint in the SimpleDemo code on BrightSpace.

Allowed cy.request() Usage (**Nowhere Else**):

- **ONLY** in beforeEach() and afterEach() hooks for setup/cleanup (if needed)
- To call the /api/reset to restore the initial state between tests.
- Refer to the SimpleDemo example for implementation.

"Why cy.request() is prohibited in test scenarios: The purpose of Assignment 3 is to test the complete user interface. Using cy.request() within scenarios bypasses the UI layer and defeats the learning objective of end-to-end testing.

*****

Every assertion in your Cypress tests MUST include a comment explaining:

1. **What the assertion is checking**
2. **How it relates to the acceptance scenario being tested**

1. A1_scenario (10%)

Scenario 1 corresponds roughly to A1's A-TEST-01.

Test the basic borrow-return cycle with two users and one book, demonstrating that:

- A borrowed book becomes unavailable to other users
- A returned book becomes available again
- Only one user can have a book at a time

Possible UI Elements to Verify:

- Book status indicators (Available/Checked Out)
- User's borrowed books list
- Due dates
- Success/error messages
- Borrow/Return buttons enabled/disabled appropriately

Testing Notes: Your test should switch between different user sessions and verify UI state changes after each action (borrow/return)

---

2. multiple_holds_queue_processing (45%)

Test the hold queue system with three users competing for the same book, demonstrating that:

- Users can place holds on unavailable books
- Hold queue follows FIFO ordering
- Notifications are sent to the correct user when book becomes available
- Only the notified user can borrow the reserved book
- Queue advances properly when reserved books are borrowed or returned

Possible UI Elements to Verify:

- Hold queue display (position in queue)
- Notification indicators/messages
- Book reservation status
- Reserved for specific user indication
- Hold/Borrow button states

Testing Notes:

- Your test should manage three different user sessions

- Verify queue ordering and notifications through the UI
- Test the complete workflow from initial borrow through multiple queue iterations

---

3. borrowing_limit_and_hold_interactions (45%)

Test the interaction between borrowing limits and holds, demonstrating that:

- Users cannot exceed the 3-book borrowing limit
- Users can place holds even when at the borrowing limit (i.e., when they already have 3 books borrowed)
- When a user at the borrowing limit returns a book, they drop below the limit
- If that user is next in the hold queue for a book, they receive a notification that the book is now available for them to borrow

Possible UI Elements to Verify:

- Borrowed book count (X/3 books borrowed)
- At-limit indicator
- Error messages for limit violations
- Hold placement while at limit
- Notification upon hold availability
- Borrowing capacity after return

Testing Notes:

- Your test should set up a user at the borrowing limit and verify constraint behaviour.
- Verify the interaction between returning books, gaining capacity, and processing held books.
- Test both positive cases (allowed actions) and negative cases (prevented actions).

Grading Breakdown

Test	**Weight**
A1_scenario	10%
multiple_holds_queue_processing	45%
borrowing_limit_and_hold_interactions	45%
Total	**100%**

To receive full grades for each of the scenarios, every assertion in your Cypress tests MUST include a comment explaining:

1. What the assertion is checking
2. How it relates to the acceptance scenario being tested

Assertions without explanatory comments will **result in deductions**. Your comments demonstrate that you understand what behaviour each assertion validates in the context of the acceptance scenario.

Deliverables (Grading Grid and README to be submitted to BrightSpace)

1. Code Repository

Branch Name: `A3-Cypress-<LastName>`

Your repository must contain:

- Complete Express.js web application
- All Cypress test files
- Custom commands (if implemented)
- Updated package.json with all dependencies
- `README.md` with setup and running instructions
- **Should not** contain package-lock.json and the node_modules folder.

Note (if you did not remove package-lock.json and node_modules folder): TAs will delete the node_modules folder and package-lock.json before running the npm install to ensure they only download the correct dependencies (express and cypress)

2. Demo Video Requirements

Record a video (approximately 10 minutes) that demonstrates:

Introduction:

- State your last name, first name, and student number clearly

Repository Setup:

- Clone or pull your repository
- Show the branch: `A3-Cypress-<LastName>`
- Show you **downloading or cloning your repository on your machine**.
- Run `npm install` to install dependencies
- Show your project structure

Code Walkthrough:

- Briefly show your Express routes/views (**if you have these**)
- Show your Cypress test file location

Test Execution (refer to SimpleDemo package.json for setting up the commands):

- Start your Express server: `npm start`
- Run Cypress in headless mode: `npm run cypress:run`
- Execute all three test scenarios
- Show each scenario passing

Test Results:

- Show the summary of all passing tests
- Verify all assertions passed

Video Quality Requirements:

- Screen recording must be clear and readable
- Audio must be clear (if you narrate)
- Keep the video under 12 minutes

3. Submission Grid

Complete the provided submission grid (to be posted separately) with:

- Your name and student number
- Repository URL
- Branch name

- Video link (YouTube, Google Drive, or OneDrive)
- Confirmation checklist

4. Running Instructions (README.md)

Include in your README:

```
## Setup Instructions
1. Install dependencies: npm install

## Running the Application
1. Start the server: npm start
2. Open browser: http://localhost:3000

## Running Cypress Tests

### Headless Mode
npm run cypress:run

## Which AI(s) were used, if they were used
```

Commit Policy for A3

You have flexibility in how you commit your work:

- You may commit in one large commit or many small commits
- Use clear, professional commit messages
- Example messages:
  - "Add Express server and routes"
  - "Implement Cypress custom commands"
  - "Add hold queue UI test"
  - "Fix borrowing limit test assertion"

All commits must be on the A3-Cypress-<LastName> branch