



SC2006 - Software Engineering Lab 4 Deliverables

Lab Group	SCEB
Team	Team 3 (HawkerGo)
Members	Dosi Veer (U2323005K)
	Razan Siraj (U2321259G)
	Chin Jiaqi (U2421026J)
	Jain Ishita Chetan (U2321668B)
	Le Nguyen Bao Huy (U2322337G)
	Liong Xun Qi (U2322609H)

Black Box Testing	3
1.1 Selected Control Class	3
1.2 Equivalence Class Testing	3
1.3 Test cases and Testing results	4
White Box Testing - updateOrderStatus	6
1. Control Flow Graph	8
2. Cyclomatic Complexity	9
3. Basis Paths	9
4. Test Cases and Testing Results	10

Black Box Testing

1.1 Selected Control Class

For our testing, we chose AuthController as the control class. This controller handles the authentication system within the application, which includes user registration and login functionalities.

During the sign-up process, users are required to provide their email address, password, and full name. Once the registration is successful, their information—along with the securely hashed password—is saved in the database. Users can subsequently log in using the same credentials they registered with.

In the login process, users enter their previously registered email and password. If the provided credentials match an existing user record (both email and hashed password), the user is successfully authenticated and redirected to the home page.

1.2 Equivalence Class Testing

Since the login and sign-up functionalities rely on distinct input values, boundary value testing is not applicable in this context. Instead, equivalence class testing is used.

- **Login Function:**

1. Valid Class: Inputs are correctly formatted and correspond to a registered user.
2. Invalid Class: Inputs are either incorrectly formatted, incomplete, or do not match any registered account.

- **Sign-Up Function:**

1. Valid Class: All required fields—email, password, and full name—are correctly filled, and the email is not already registered.
2. Invalid Class: The Inputs are missing, the email format is incorrect, or the email already exists in the system.

1.3. Test Cases and Testing Results

1.3.1 Sign up

Input parameters:

1. Email
2. Password
3. Full Name
4. User

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
Signup-01	(Valid) Email: "testuser@gmail.com" (Valid) Password: "testpassword" (Valid) Full Name: "Test Name" User: Customer	Account created! Please login.	Account created! Please login.	Pass
Signup-02	(Invalid) Email: "testuser@gmail.com" (Valid) Password: "testpassword" (Valid) Full Name: "Test Name" User: Customer	Error: "User already exists"	Error: "User already exists."	Pass
Signup-03	(Invalid) Email: "" (Valid) Password: "testpassword" (Valid) Full Name: "Test Name" User:Customer	Error: "Please fill in all fields"	Error: "Please fill in all fields"	Pass
Signup-04	(Invalid) Email: "testuseratgmaildotcom" (Valid) Password: "testpassword" (Valid) Full Name: "Test Name" User:Customer	Error: Registration failed	Error: Registration failed	Pass
Signup-05	(Valid) Email: "testuser@gmail.com" (Invalid) Password: "" (Valid) Full Name: "Test Name" User:Customer	Error: "Please fill in all fields"	Error: "Please fill in all fields"	Pass
Signup-06	(Valid) Email: "testuser@gmail.com" (Invalid) Password: "1234" (Valid) Full Name: "Test Name" User: Customer	Error: Registration failed	Error: Registration failed	Pass
Signup-07	(Valid) Email: "testuser@gmail.com" (Valid) Password: "testpassword" (Invalid) Full Name: "" User:Customer	Error: "Please fill in all fields"	Error: "Please fill in all fields"	Pass

Signup-08	(Invalid) Email: "testuser@gmail.com" (Valid) Password: "testpassword" (Valid) Full Name: "Test Name" User: Stall Owner	Error: "User already exists"	Error: "User already exists"	Pass
-----------	--	------------------------------	------------------------------	------

1.3.2 Login

Input parameters:

Email

Password

Test Case Name	Test Input	Expected output	Actual output	Test Result
Login-01	(Valid) Email: "testuser@gmail.com" (Valid) Password: "testpassword"	Login Success	Login Success	Pass
Login-02	Valid Email: "testuser@gmail.com" (Invalid) Password: "wrongpassword"	Error: "Invalid credentials"	Error: "Invalid credentials"	Pass
Login-03	(Valid) Email: "testuser@gmail.com" (Invalid) Password: ""	Error: "Please fill in all fields"	Error: "Please fill in all fields"	Pass
Login-04	(Invalid) Email: "fakeuser@gmail.com" (Valid) Password: "testpassword"	Error: "Invalid credentials"	Error: "Invalid credentials"	Pass
Login-05	(Invalid) Email: "" (Valid) Password: "testpassword"	Error: "Please fill in all fields"	Error: "Please fill in all fields"	Pass
Login-06	(Invalid) Email: "testuseratgmaildotcom" (Valid) Password: "testpassword"	Error: Login Failed	Error: Login Failed	Pass

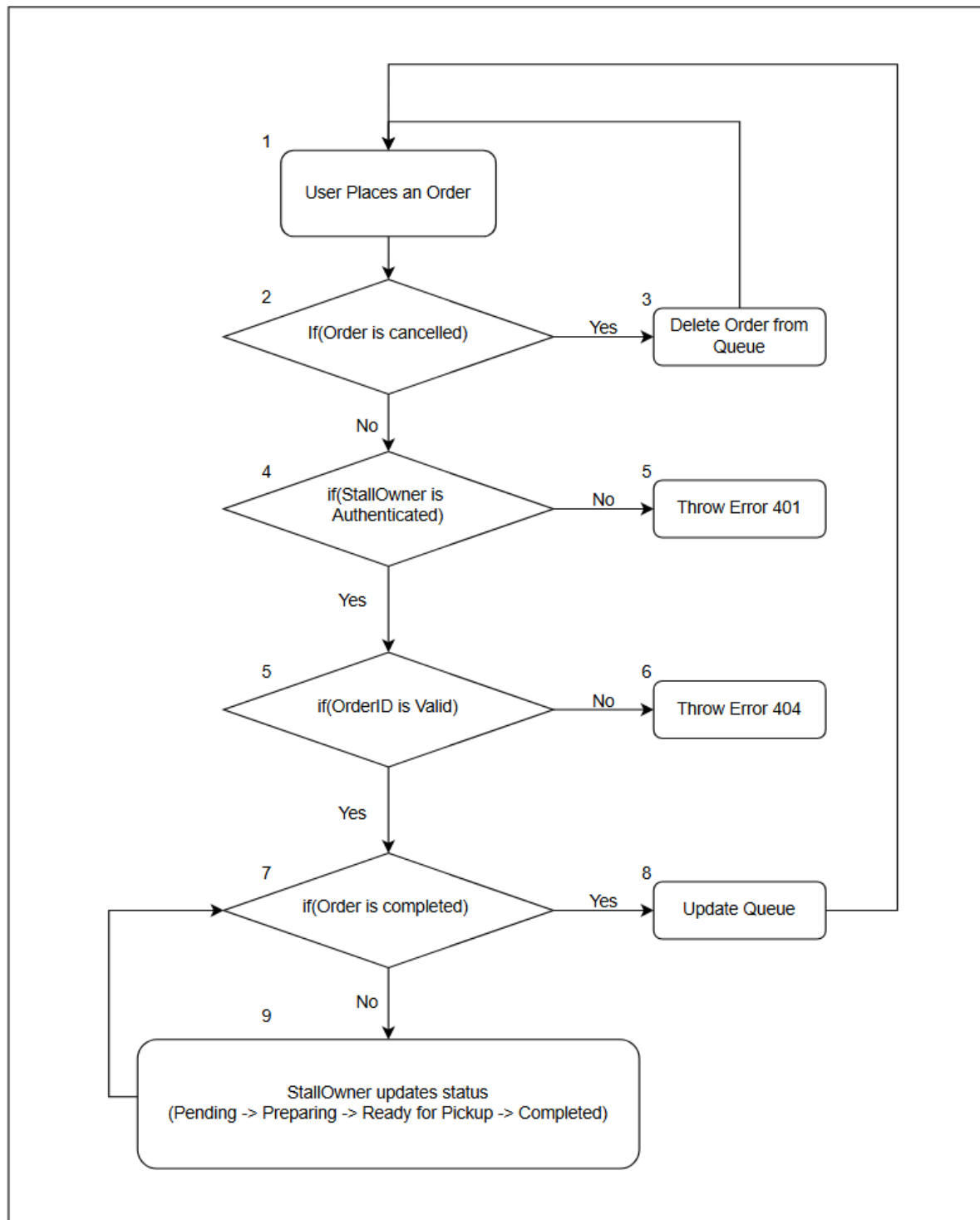
White Box Testing - updateOrderStatus

```
350 describe('PUT /api/orders/:orderId/status (updateOrderStatus)', () => {
351   it('should allow stall owner to update status (e.g., to preparing)', async () => {
352     await Order.findByIdAndUpdate(testOrder._id, { status: 'pending' }); // Ensure starting state
353     const statusUpdate = { status: 'preparing' };
354     const res = await request(app)
355       .put(`/api/orders/${testOrder._id}/status`)
356       .set('x-auth-token', tokenOwner)
357       .send(statusUpdate);
358
359     expect(res.statusCode).toBe(200);
360     expect(res.body._id.toString()).toBe(testOrder._id.toString());
361     expect(res.body.status).toBe('preparing');
362   });
363
364   it('should update queue currentNumber when order is COMPLETED', async () => {
365     // Set order to 'ready' and queue number
366     const testQueueNum = 55;
367     await Order.findByIdAndUpdate(testOrder._id, { status: 'ready', queueNumber: testQueueNum });
368     // Ensure queue currentNumber is behind
369     await Queue.findByIdAndUpdate(testQueue._id, { currentNumber: testQueueNum - 1 });
370
371     const statusUpdate = { status: 'completed' };
372     const res = await request(app)
373       .put(`/api/orders/${testOrder._id}/status`)
374       .set('x-auth-token', tokenOwner)
375       .send(statusUpdate);
376
377     expect(res.statusCode).toBe(200);
378     expect(res.body.status).toBe('completed');
379
380     const updatedQueue = await Queue.findById(testQueue._id);
381     // Controller logic advances currentNumber TO the completed order's number
382     expect(updatedQueue.currentNumber).toBe(testQueueNum);
383   });
384
385   it('should NOT update queue currentNumber if status is not completed/cancelled', async () => {
386     const initialQueue = await Queue.findById(testQueue._id);
387     const statusUpdate = { status: 'ready' }; // Update to something other than completed/cancelled
388     const res = await request(app)
389       .put(`/api/orders/${testOrder._id}/status`)
390       .set('x-auth-token', tokenOwner)
391       .send(statusUpdate);
392
393     expect(res.statusCode).toBe(200);
394     expect(res.body.status).toBe('ready');
395
396     const finalQueue = await Queue.findById(testQueue._id);
397     expect(finalQueue.currentNumber).toBe(initialQueue.currentNumber); // Should not change
398   });
399
400   it('should return 401 if user is not the stall owner', async () => {
401     const statusUpdate = { status: 'ready' };
402     const res = await request(app)
403       .put(`/api/orders/${testOrder._id}/status`)
404       .set('x-auth-token', tokenCustomer) // Use customer token
405       .send(statusUpdate);
406     expect(res.statusCode).toBe(401);
407   });
408
409   it('should return 404 if order ID is invalid', async () => {
410     const invalidMongoId = new mongoose.Types.ObjectId().toString();
411     const statusUpdate = { status: 'ready' };
412     const res = await request(app)
413       .put(`/api/orders/${invalidMongoId}/status`)
414       .set('x-auth-token', tokenOwner)
415       .send(statusUpdate);
416     expect(res.statusCode).toBe(404);
417   });
418
419   it('should return 401 if not authenticated', async () => {
420     const statusUpdate = { status: 'ready' };
421     const res = await request(app)
422       .put(`/api/orders/${testOrder._id}/status`)
423       .send(statusUpdate); // No token
424     expect(res.statusCode).toBe(401);
425   });
426 });
```

Method selected for white box testing: PUT /api/orders/:orderId/status (updateOrderStatus)

The `updateOrderStatus` method updates the status of an order that is already placed by a customer, potentially affecting the current number in the queue associated with the order. This includes multiple scenarios such as order status updates to various states and validation of user authentication and authorization.

1. Control Flow Graph



2. Cyclomatic Complexity

Cyclomatic Complexity (CC) is calculated as the number of binary decision points + 1.

From the control flow diagram, there are four decision points:

1. Order cancellation check
2. User authentication check
3. Order ID validity check
4. Order completion check

Thus, Cyclomatic Complexity (CC) = 4 + 1 = 5

3. Basis Paths

The following are the independent basis paths derived from the control flow graph:

- Path #1 (Baseline): Authenticated stall owner successfully updates status to 'preparing'.
- Path #2: Status updated to 'completed', triggering queue currentNumber update.
- Path #3: Status updated to a non-terminal state like 'ready', queue remains unchanged.
- Path #4: User is authenticated but not a stall owner — receives 401 Unauthorized.
- Path #5: Authenticated user provides an invalid order ID — receives 404.
- Path #6: Request made without authentication — receives 401 Unauthorized.

4. Test Cases and Testing Results

Input Parameters:

1. status
2. orderId (Automatically passed from URL)
3. user authentication token (passed in request headers)

Test Case Name	Test Input	Expected Output	Actual Output	Test Result
OrderStatus-01	status = "preparing", valid orderId, authenticated stall owner	Status update successful, status = "preparing"	Status update successful, status = "preparing"	Pass
OrderStatus-02	status = "completed", valid orderId, queueNumber = 55, queue currentNumber = 54	Queue currentNumber updated to 55, status = "completed"	Queue currentNumber updated to 55, status = "completed"	Pass
OrderStatus-03	status = "ready", valid orderId, queue currentNumber should not change	Queue currentNumber unchanged, status = "ready"	Queue currentNumber unchanged, status = "ready"	Pass
OrderStatus-04	status = "ready", valid orderId, authenticated non-stall owner	Error 401: Unauthorized user	Error 401: Unauthorized user	Pass
OrderStatus-05	status = "ready", invalid orderId	Error 404: Order ID invalid	Error 404: Order ID invalid	Pass
OrderStatus-06	status = "ready", valid orderId, no authentication token	Error 401: Not authenticated	Error 401: Not authenticated	Pass

PASS tests/controllers/queueController.test.js

Order Controller Tests (Using Test DB & Selective Cleanup)

POST /api/orders (createOrder)

- ✓ should create a new order successfully (79 ms)
- ✓ should create a loyalty record if none exists (51 ms)
- ✓ should update an existing loyalty record (56 ms)
- ✓ should return 404 if stallId does not exist (11 ms)
- ✓ should return 400 if queue is not active (32 ms)
- ✓ should return 401 if user is not authenticated (8 ms)

GET /api/orders/user (getUserOrders)

- ✓ should return orders placed by the authenticated user (25 ms)
- ✓ should return an empty array if the authenticated user has no orders (14 ms)
- ✓ should return 401 if not authenticated (4 ms)

GET /api/orders/stall/:stallId (getStallOrders)

- ✓ should return non-completed orders for the stall owner (28 ms)
- ✓ should return 401 if user is not the stall owner (7 ms)
- ✓ should return 401 if stall does not exist or owner mismatch (6 ms)
- ✓ should return 401 if not authenticated (2 ms)

PUT /api/orders/:orderId/status (updateOrderStatus)

- ✓ should allow stall owner to update status (e.g., to preparing) (31 ms)
- ✓ should update queue currentNumber when order is COMPLETED (49 ms)
- ✓ should NOT update queue currentNumber if status is not completed/cancelled (38 ms)
- ✓ should return 401 if user is not the stall owner (12 ms)
- ✓ should return 404 if order ID is invalid (8 ms)
- ✓ should return 401 if not authenticated (2 ms)

PUT /api/orders/:orderId/cancel (cancelOrder)

- ✓ should allow customer to cancel a PENDING order (37 ms)
- ✓ should NOT allow customer to cancel if status is NOT pending (38 ms)
- ✓ should return 401 if user did not place the order (22 ms)
- ✓ should return 404 if order ID is invalid (21 ms)
- ✓ should return 401 if not authenticated (19 ms)

Test Suites: 1 passed, 1 total

Tests: 24 passed, 24 total

Snapshots: 0 total

Time: 2.347 s, estimated 30 s