Ho Chi Minh University of Science

Department of Information Technology

Software Testing

Course:	**Introduction to Software Engineering**
Instructor:	**Mr. Truong Phuoc Loc**
Class:	**23CLC03**
Group:	**07**
Students' name:	**Le Anh Duy – 23127011**
	Tran Gia Huy – 23127199
	Nguyen Thanh Luan – 23127296
	Nguyen Thanh Tien – 23127539

Ho Chi Minh City, December 2025

Table of Contents

1 Member Contribution Assessment

ID	Name	Contribution (%)	Signature
23127011	Le Anh Duy	100%	duy
23127199	Tran Gia Huy	100%	Huy
23127296	Nguyen Thanh Luan	100%	Luan
23127539	Nguyen Thanh Tien	100%	[illegible]

2 Test plan

2.1 Scope

2.1.1 In Scope

The manga-sharing web system consists of three subsystems:

Reader Subsystem

- Account registration/login (security, encryption).
- Read chapters: display pages, choose translation language, translate per page in “real time”.
- Track reading progress: current chapter, last page read, translation status (synchronized).
- Commenting and rating.
- Share chapters to external platforms.
- Payment: pay to read manga using multiple methods (e.g., credit cards, ZaloPay), and grant access after successful payment.

Uploader Subsystem

- Users submit a request to become an uploader (Reader → Request uploader).
- Uploaders upload manga chapters and provide metadata: chapter title, manga title, genre, number of pages, original language, and a short summary.
- Manage uploaded content: edit metadata and update chapters before submitting for review.
- Submit chapters for admin approval.
- Track chapter status: Draft / Submitted / Approved / Rejected / Published.
- View basic statistics such as views and ratings to improve content quality.

Admin Subsystem

- User management: assign roles (Admin/Uploader/Reader), approve uploader requests, lock/suspend accounts.
- Content management: review chapters, approve/reject, publish/unpublish (if applicable).
- Highlighted/Recommended: mark chapters for priority display, set an end date (default 7 days), adjust the end date, and automatically expire the priority.
- Third party translation service: configure the integration and control how translations are stored and displayed.

2.1.2 Out of Scope

- Evaluating the "semantic correctness/context accuracy" of translations (testing focuses on: translation exists, correct format, correct display, and error/latency handling).
- Internal failures of third-party Payment/Translation providers (testing focuses only on how our system behaves when APIs fail/timeout/rate-limit).

2.2 Quality Objectives

- Functional correctness: business workflows work correctly for Reader/Uploader/Admin.
- Security: prevent permission bypass; mitigate basic XSS/SQLi/CSRF risks; protect user data.
- Reliability: no loss of reading progress; upload/approval workflow maintains correct states.
- Usability: smooth CSR experience with clear loading/error states.
- Performance: per-page translation does not freeze the UI; key APIs respond within acceptable limits.

2.3 Test Item

2.3.1 Software Components (Functions/Modules)

Cross subsystem

Frontend

Function (Core)	Techniques	Expected Functionality	Module
FormForgotPassword()	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Render the forgot-password form, validate user input, submit a password reset request to the backend, and navigate the user to the OTP verification flow based on the response outcome.	Account
FormLogin()	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Render the login form, validate inputs, submit login requests, handle success/failure outcomes, and redirect users to the appropriate area based on their role.	Account
FormRegister()	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Render the registration form, validate user inputs and required consent, submit a registration request, and guide the user to the verification step based on the outcome.	Account
FormResetPassword()	Static: Walkthroughs; Dynamic	Render the reset-password	Account

	– Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	form, validate new password rules and confirmation matching, submit the reset request, and redirect the user to login upon successful completion.	
ProfileDetailPage()	Static: Walkthroughs; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Display the user profile editing page, prefill existing user data, support avatar update, validate editable fields, submit profile updates, and navigate based on the update outcome.	Profile

Backend

Function (Core)	Techniques	Expected Functionality	Module
register(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Reject if email exists; delete expired OTP; reject if OTP already sent; generate OTP; store OTP+email; create verified_otp_token (JWT) and set cookie; send OTP email; return “enter OTP” response.	Account
registerVerify(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic –	Verify OTP with email; if valid then hash password, generate username, insert account, delete OTP, clear	Account

	Experience-based: Error Guessing. Static: Walkthroughs.	cookie, return success; on error clear cookie and cleanup OTP.	
login(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Validate email exists; validate password; generate accessToken with role/id/email; set accessToken cookie with expiry (3d or 1d); return success + role.	Account
forgotPassword(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Check email exists; delete expired OTP; reject if OTP already sent; generate/store OTP; create verified_otp_token cookie; send OTP email; return “enter OTP” response.	Account
forgotPasswordVerify(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Verify OTP for email; if valid return success prompting new password; else return OTP error.	Account
resetPassword(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Reset password: hash new password; update DB password; delete OTP record; clear cookie; return success.	Account

checkUser(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Read accessToken from cookies; if missing then return error. Extract user info from DB, if succeed return success with user info.	Authorization
logout(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Clear accessToken cookie and return.	Authorization
list(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Fetch all coin packages from DB and return success and coin packages.	Coin System
detail(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	fetch order/package detail via id, return success with detail; on error return message.	Coin System
paymentZaloPay(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Create a ZaloPay payment request, call ZaloPay create endpoint via Axios;return success and provide paymentUrl; otherwise return error.	Payment

paymentZaloPayResult(req, res)	Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Handle ZaloPay callback: build order record and persist via orderModel.createOrder; set return_code=1 on success; on exception set return_code=0 to allow ZaloPay retry (max 3).	Payment
profile(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing. Dynamic – Experience-based: Error Guessing. Static: Walkthroughs.	Update user profile: Check if username already exists; if exists and belongs to another user then return error. Otherwise update profile in DB for current user and return success; return error if update fails.	Profile

Reader subsystem

Frontend

Function (Core)	Techniques	Expected Functionality	Module
RegisterUploaderPage()	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing, Dynamic – Experience-based: Error Guessing, Exploratory Testing	Display the uploader registration page, present platform rules, validate prerequisites and consent, collect the application reason, submit the request, and navigate based on the submission outcome.	Upload Register

Backend

Function (Core)	Techniques	Expected Functionality	Module
registerUploader(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing; Dynamic – Experience-based: Error Guessing; Static: Walkthroughs	Create uploader registration request: return success message "waiting"; return error if DB insert fails.	Upload Register

Uploader subsystem

Frontend

Function (Core)	Techniques	Expected Functionality	Module
UploadMangaPage()	Static: Walkthroughs; Dynamic – Specification-based: Use Case Testing, Dynamic – Experience-based: Error Guessing, Exploratory Testing	Provide an uploader-only interface to upload a new manga or add a new chapter, collect required inputs/files, submit the upload request, and guide the user based on outcomes.	Upload

Backend

Function (Core)	Techniques	Expected Functionality	Module
uploadManga(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing; Dynamic – Experience-based: Error Guessing; Static: Walkthroughs	Upload manga content (new manga or existing manga). Validate uploader authentication. Return success; on error return 4xx/5xx with message.	Upload Manga
getMyMangas(req, res)	Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing; Dynamic – Experience-based: Error Guessing; Static: Walkthroughs	Require authenticated uploader. Fetch mangas uploaded by the current uploader and return list; handle server errors.	Upload Manga

Admin subsystem

Frontend

Function (Core)	Techniques	Expected Functionality	Module
GenreCreateForm()	Static: Walkthroughs; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing	Render the genre creation form, validate required fields, collect editor content, generate a slug, submit the create request to the backend, and provide user feedback and navigation actions.	Manga Management
ManageMangaPage()	Static: Walkthroughs; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Display the admin manga management view, load and present manga records, support filtering/search/selection, and allow moderation actions such as approving or hiding manga.	Manga Management
EditUserPage()	Static: Walkthroughs; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing, Exploratory Testing	Display the admin user-edit page, load user details, show read-only profile information, allow updating role and account status, submit updates, and refresh/navigate based on outcomes.	User Management

Backend

Function (Core)	Techniques	Expected Functionality	Module
getListManga(req, res)	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing	Retrieve and return a list of mangas with uploader information for admin content management, with appropriate ordering and basic error handling.	Manga Managament
approveManga(req, res)	Static: Walkthroughs; Dynamic – Specification-based: State Transitioning, Use Case Testing; Dynamic – Experience-based: Error Guessing	Approve a manga by updating its status to an approved/active state and return the operation result.	Manga Management
rejectManga(req, res)	Static: Walkthroughs; Dynamic – Specification-based: State Transitioning, Use Case Testing; Dynamic – Experience-based: Error Guessing	Reject or hide a manga by updating its status to a rejected/inactive state and return the operation result.	Manga Management
list(req, res)	Static: Walkthroughs, Technical Reviews; Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing	Retrieve and return the full list of users for admin management, with error handling.	User Management
detail(req, res)	Static: Walkthroughs; Dynamic – Specification-based: Equivalence	Retrieve and return details of a specific user if they exist; otherwise return a not-found	User Management

	Partitioning, Use Case Testing; Dynamic – Experience-based: Error Guessing	result.	
update(req, res)	Static: Walkthroughs; Dynamic – Specification-based: Equivalence Partitioning, Boundary Value Analysis, Use Case Testing; Dynamic – Experience-based: Error Guessing	Update a user's information based on admin input and return the operation result.	User Management

2.3.2 External Integrations

Service (Core)	Subsystem	Techniques	Expected Functionality
Translation Service	Reader, Uploader	Dynamic – Specification-based: Use Case Testing; Dynamic – Experience-based: Error Guessing	Responsible for sending manga pages to Google API, validating the returned results, storing translations for reuse, and delivering translated text to the reader while handling timeouts, rate limits, and other failures.
Payment Service	Reader, Uploader	Dynamic – Specification-based:	Provides available

		Use Case Testing; Dynamic – Experience-based: Error Guessing	purchase options, initiates payment sessions with external gateways, processes callbacks/webhooks securely, updates transaction status.

2.4 Test Levels

2.4.1 Unit Testing

- Backend: authentication & OTP flows, role-based authorization checks, uploader registration workflow, manga upload processing (cover + archive parsing + page ordering), admin moderation actions (approve/reject with status transition rules), payment webhook handling (verification + idempotency).
- Frontend: route guards, state updates (progress), error states.

2.4.2 Integration Testing

- API ↔ Translation Service: schema + timeout + retry/backoff.
- API ↔ Payment: sandbox create payment + webhook verification + replay.

2.4.3 System / E2E Testing (core scenarios)

Scenario Reader

- Register/Login -> Open chapter -> Translate 1–N pages -> Save progress -> Reload -> Resume
- Comment & rating -> verify public display
- Payment success -> unlock content -> read

Scenario Uploader

- Reader request uploader -> uploader -> admin approve -> uploader can upload manga.
- Uploader create draft -> upload pages -> edit metadata -> submit -> wait for approval.
- If rejected, then resubmit.

Scenario Admin

- Admin approve/reject/publish chapter -> reader then can view the chapter.
- Set highlighted/recommended (default 7 days) -> set date -> outdated mangas is not featured.
- Lock/suspend user -> user is locked, cannot read.

2.5 Test Completeness

- Pass all test cases that are created.
- Traceability ensure main functionalities (Reader/Uploader/Admin) all have suitable test cases and results.

2.6 Testing Tools

- Unit/Integration: Jest, Supertest
- Frontend tests: Testing Library
- E2E: Playwright
- API manual: Postman
- Performance: JMeter
- Security: OWASP ZAP

2.7 Test Environments

- Staging: Next.js CSR + Express + PostgreSQL
- Translation: Google Nano Banana
- Payment: ZaloPay
- Test accounts: admin/uploader/reader

- Browsers: Chrome/Edge/Firefox

3 Test cases

3.1 List of test cases

Seq	Test case	Target	Description
1	Upload New Comic (Pending Approval)	Comic Management (Uploader)	Verify that when an uploader posts a new comic, the system sets the status to "Pending" for approval.
2	Admin Approves New Comic	Comic Management (Admin)	Verify that the Admin can approve a pending comic, changing its status to "Active" and making it visible publicly.
3	Upload New Chapter (Auto-Public)	Comic Management (Uploader)	Verify that uploading a new chapter to an already "Active" comic publishes the content immediately without requiring re-approval.
4	Automated Comic Translation (Offline Trigger)	Translation Service	Verify the integration of the external translation API, ensuring English files are translated to Vietnamese in the background.

5	Translation Service Unavailable	Translation Service	Verify system fail-safe behavior: if the translation API fails/disconnects, the upload succeeds with the original content instead of crashing.
6	Deposit Via ZaloPay	Payment System	Verify the user can successfully top up their wallet (Coin) using the ZaloPay gateway (Sandbox environment).
7	Unlock Premium Chapter Success	Payment System	Verify that a user with sufficient balance can successfully unlock a premium chapter, deducting coins and revealing content.
8	Unlock Premium Insufficient Fund	Payment System	Verify that the system prevents access, shows an error message, and suggests depositing when the user lacks sufficient funds.
9	Uploader Reward System	Reward System	Verify that the Uploader automatically receives a Coin reward after a successful chapter upload.
10	Security Access Locked Content	Security	Verify that unauthorized users cannot access premium

			content via direct URL manipulation (URL Hacking)
11	Unauthorized User Attempt Upload	Security	Verify Role-Based Access Control: Regular users (Readers) cannot access upload interfaces even if they know the direct URL.
12	Admin Reject Comic	Comic Management (Admin)	Verify that the Admin can reject a violating comic, setting the status to "Rejected" and notifying the uploader

3.2 Test case specifications

Group 1: Comic & Chapter Management (Upload/Approval)

3.2.1 Test Case 1

Test Case	**Upload New Comic (Pending Approval)**
Related Use Case	Upload Comic (Uploader)
Context	An Authorized Uploader wants to post a completely new comic series.

Input Data	- Name: "New Legend" - Cover Image: cover.jpg - Author: John Doe - Description: "An adventure story..."
Expected Output	- System notifies "Upload successful, please wait for approval". - Comic status in DB/Dashboard is: Pending. - Comic is not visible on the homepage (Regular users cannot see it).
Test Steps	1. Log in as Authorized Uploader. 2. Go to "Comic Management" menu > Select "Add New Comic". 3. Enter full details (Name, Author, Cover, Description). 4. Click "Upload" button. 5. Check "My Comics" list and the homepage (using Incognito mode).
Actual Output	Matches expected output
Result	Passed

3.2.2 Test Case 2

Test Case	**Admin Approves New Comic**
Related Use Case	Approve Comic (Admin)

Context	A comic is currently in "Pending" status.
Input Data	Comic ID: "New Legend" (Pending).
Expected Output	- Comic status changes to: Active. - Comic appears publicly on the homepage and search results. - Uploader receives a notification that the comic has been approved.
Test Steps	1. Log in as Admin. 2. Go to "Comic Approval" Dashboard. 3. Select "New Legend" and view details. 4. Click "Approve" button. 5. Open the homepage in Guest mode to verify visibility.
Actual Output	Matches expected output
Result	Passed / Failed

3.2.3 Test Case 3

Test Case	**Upload New Chapter (Auto-Public)**
Related Use Case	Upload Chapter (Uploader)

Context	The comic series is already approved (Active). Uploader adds a new Chapter.
Input Data	- Comic: "New Legend" - File: chap1.zip - Order: Chapter 1
Expected Output	- Chapter is uploaded successfully. - Chapter status is Active immediately (skips approval step). - Readers can view the content immediately.
Test Steps	1. Log in as Uploader. 2. Select the Active comic. 3. Select "Add Chapter" > Upload content image file. 4. Click "Save/Post". 5. Use a Reader account to try reading the newly posted chapter.
Actual Output	Matches expected output
Result	Passed / Failed

Group 2: Automated Translation Features

3.2.4 Test Case 4

Test Case	Automated Comic Translation (Offline Trigger)
Related Use Case	Automated Translation
Context	Uploader posts a comic with English source. System integrates with an external translation service API.
Input Data	- Comic File: comic_english.zip - Source Language: English - Target Language: Vietnamese
Expected Output	- System receives file and pushes it to the offline translation queue. - After the service finishes, the chapter has a Vietnamese version (or Vietnamese text overlay). - Display status: "Translated".
Test Steps	1. Upload Chapter with language flag set to "English". 2. Complete upload. 3. Wait for background service processing (monitor logs or status notification). 4. Open the chapter to verify if the content has been converted to Vietnamese.

Actual Output	Matches expected output
Result	Passed

3.2.5 Test Case 5

Test Case	**Translation Service Unavailable**
Related Use Case	Automated Translation
Context	External translation service is disconnected or returning errors.
Input Data	- English comic file. - Simulated Service: Disconnected/Error 500.
Expected Output	- Upload process does not crash. - Chapter is still created with original content (English). - System logs the translation error for Admin review.
Test Steps	1. Disconnect Translation API (or use Mock API returning error). 2. Perform English comic upload. 3. Verify if the comic creation is successful. 4. Check displayed content (must be original English, not a blank page).

Actual Output	Matches expected output
Result	Passed

Group 3: Payment & Currency (ZaloPay & Coin)

3.2.6 Test Case 6

Test Case	**Deposit via ZaloPay**
Related Use Case	Deposit Coin (User)
Context	User wants to top up wallet. Current balance: 0 Coin.
Input Data	- Amount: 20,000 VND - Method: ZaloPay QR
Expected Output	- Redirect to ZaloPay payment gateway. - After successful payment, redirect back to the website. - Wallet balance increases by 20 Coins (assuming 1:1 ratio). - Transaction history: "Success".
Test Steps	1. Log in as Regular User. 2. Go to "Wallet" > "Deposit" > Select 20k package via ZaloPay. 3. Perform payment (Sandbox/Test environment).

	4. Return to website, refresh page, and check coin balance.
Actual Output	After successful payment, ZaloPay not redirect to website
Result	Failed

3.2.7 Test Case 7

Test Case	**Unlock Premium Chapter Success**
Related Use Case	Purchase Comic (User)
Context	VIP Chapter costs 5 Coins. User has 50 Coins.
Input Data	- User Balance: 50 Coins - Chapter Price: 5 Coins
Expected Output	- Deduct 5 Coins from wallet (remaining 45 Coins). - Chapter unlocks immediately. - User can view the image content.
Test Steps	1. Access Chapter with lock icon (VIP). 2. Click "Unlock Now - 5 Coins". 3. Confirm on popup. 4. Check comic content display and wallet balance afterwards.

Actual Output	Matches expected output
Result	Passed

3.2.8 Test Case 8

Test Case	Unlock Premium Insufficient Fund
Related Use Case	Purchase Comic (User)
Context	VIP Chapter costs 5 Coins. User only has 2 Coins.
Input Data	- User Balance: 2 Coins - Chapter Price: 5 Coins
Expected Output	- System displays error: "Insufficient balance". - Show button/link redirecting to Deposit page. - Chapter remains locked.
Test Steps	1. Access VIP Chapter. 2. Click "Unlock". 3. Observe error message and system behavior.
Actual Output	Matches expected output

Result	Passed

3.2.9 Test Case 9

Test Case	**Uploader Reward System**
Related Use Case	Receive Reward (Uploader)
Context	Mechanism: Reward 2 Coins for every successful chapter upload.
Input Data	- Initial balance: 10 Coins - Action: Successfully upload 1 chapter.
Expected Output	- Uploader balance increases to 12 Coins. - Balance history log: "Reward for uploading comic X".
Test Steps	1. Check current Uploader balance. 2. Perform new chapter upload and wait for completion. 3. Go to wallet management page to verify coin balance.
Actual Output	Matches expected output
Result	Passed

Group 4: Security & Roles

3.2.10 Test Case 10

Test Case	Security Access Locked Content
Test Case Name	Unauthorized Access to Paid Content (URL Hacking)
Related Use Case	System Security
Context	User has not purchased Chapter ID 123 (Premium). User tries to guess API URL to fetch images.
Input Data	Direct URL: website.com/api/read/123 or unsigned CDN image URL.
Expected Output	- System blocks access. - Returns 403 Forbidden error or redirects to payment page. - Comic image content is NOT revealed.
Test Steps	1. Get direct link of a premium chapter (from network tab of a purchased user - if advanced testing, or guess the link). 2. Paste into browser of a User who hasn't purchased it.

	3. Check response.
Actual Output	Matches expected output
Result	Passed

3.2.11 Test Case 11

Test Case	**Unauthorized User Attempt Upload**
Related Use Case	User Roles (RBAC)
Context	Regular Reader account (not authorized by Admin as Uploader).
Input Data	Access URL: website.com/uploader/create
Expected Output	- Website displays "Access Denied" or "You do not have permission". - Upload form is not displayed. - Redirect to homepage or login page.
Test Steps	1. Log in as Reader. 2. Try to find Upload button (Expected: None). 3. Type upload page URL directly into address bar. 4. Check display result.

Actual Output	Matches expected output
Result	Passed

3.2.12 Test Case 12

Test Case	**Admin Reject Comic**
Related Use Case	Approve Comic (Admin)
Context	Newly uploaded comic violates rules (inappropriate content), status Pending.
Input Data	- Comic: "Violating Comic" - Action: Reject - Reason: "Inappropriate Content"
Expected Output	- Comic status: Rejected. - Comic is not displayed externally. - Uploader receives notification with rejection reason.
Test Steps	1. Admin goes to Approval list. 2. Select comic to reject. 3. Click "Reject" and enter reason. 4. Log in as Uploader to check status and notification.
Actual Output	Matches expected output

Result	Passed