Test Document

for

Re_Store

Version 1.0

Prepared by

Group:4 **Group Name: Divide and Conquer**

Bharatula Anirudh Srivatsa	230290	bharatula23@iitk.ac.in
Voora Rakesh	231174	vrakesh23@iitk.ac.in
Soma Koushik	231018	koushiks23@iitk.ac.in
Sanjay Raghav Vangala	230916	sanjayra23@iitk.ac.in
Yashwanth Reddy Junutula	231194	yashwanth23@iitk.ac.in
Saatvik Gundapaneni	230428	gundapanen23@iitk.ac.in
Vempati Prem Santhosh	231137	psanthosh23@iitk.ac.in
Ayush Yadav	230272	ayushydv23@iitk.ac.in
Meghana Kadari	230512	kmeghana23@iitk.ac.in
Chapati Venkata Pranaya	230324	chapative23@iitk.ac.in

Course: CS253
Mentor TA: Jeswaanth Gogula
Date of Submission: 6th April 2025

CONTENTS

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.00	Bharatula Anirudh Srivatsa Voora Rakesh Soma Koushik Sanjay Raghav Vangala Yashwanth Reddy Junutula Saatvik Gundapaneni Vempati Prem Santhosh Ayush Yadav Meghana Kadari Chapati Venkata Pranaya	The First version of the Test Document	06/04/2025

1 Introduction

☐ **Testing Strategy:**

In our Project we have done both manual and automated testing to check the robustness and payload of our software. We have used Postman software to send HTTP /POST and /GET requests to the backend server for manual testing. Using Postman software we were able to send manual requests to the backend very similar to the request we send from the frontend, requests to different endpoints/backend routes and tallied the received responses against the expected response.

☐ **Testing Timeline:**

We conducted our software testing in three distinct phases. Initially, we rigorously tested each component individually before integrating them into the complete codebase. This phase ensured that each module functioned correctly in isolation.

In the second phase, which ran concurrently with the implementation process, developers tested their own branches thoroughly before merging them into the main codebase. Manual integration and system testing were carried out to verify the seamless combination of components and to identify any issues that arose during this stage.

The third phase involved the development of a custom testing mechanism tailored to the platform. This solution incorporated various testing methodologies to ensure comprehensive evaluation. The output from these tests was carefully reviewed through logs and runtime behavior to verify correctness and reliability.

Throughout the second phase, particular attention was given to challenges encountered during the integration process. Key features and security enhancements were thoroughly tested through pull requests. Before merging these pull requests, extensive testing and conflict resolution were performed to ensure the stability and integrity of the codebase.

☐ **Testers and the testing approach:**
We have divided the team in two sub groups with the responsibility of testing frontend/Integration testing and the backend testing. There were also two members who were making sure that the whole application was being tested. We adopted a unique and good approach that the testing of the backend of the web app was done by the students who worked on the front end, and vice versa

☐ **Coverage Criteria:**
In manual testing, we consider the number of lines of code tested by reviewing intermediate values through printing and assessing whether the required functionalities are fulfilled by our software. Additionally, we incorporate Branch Coverage criteria during testing to ensure a comprehensive evaluation. Recognizing the impracticality of testing all potential errors, we set a target coverage of 95% to strike a balance between thoroughness and feasibility.

☐ **Tools Used:**

- Unit Testing: Postman
- System Testing: ApacheBench, a single-threaded command line computer program used for benchmarking HTTP web servers, was used. ApacheBench (ab) measures the performance of a web server by inundating it with HTTP requests and recording metrics for latency and success.

2 Unit Testing

1. authController

The test was conducted to verify the authentication system for both **Buyer** and **Seller**, ensuring that **Signup**, **Login**, and **Access Control** work correctly under valid, invalid, and empty input conditions.

The tests conducted passed successfully

Unit Details: Functions: signup, login, forgotPassword, resetPassword, updatePassword
Test Owner: Yashwanth Reddy
Test Date: 03/04/2025
Test Results: Passed. Under right inputs, JWT token was generated whereas under wrong and empty inputs exceptions were raised.

a. POST /api/v1/users/signup
This test ensures that the endpoint correctly handles user registration. It verifies that a JWT token is returned upon successful signup and that the input is processed correctly.

Inputs: {email:test@example.com, name=Test User, password="Test@123"}
Output: JWT Token: eyJhbGciOiJIUzI1NiIs... [PASSED]

Test Results:
created a new user and returned token (4 ms)
handled validation errors (39 ms)

b. POST /api/v1/users/login
This test ensures that the endpoint correctly handles user authentication. It verifies appropriate responses for valid logins, incorrect passwords, non-existent users, and unauthorized access due to insufficient privileges.

Inputs: {email:test@example.com, name=Test User, password="Test123"}
Output: Password not valid [PASSED]

Inputs: {email:test@example.com, name=Test, password="Test@123"}
Output: User not found [PASSED]

Inputs: {email:admin@example.com, password="Admin@123"}
Output: JWT Token: eyJhbGciOiJIUzI1NiIs... [PASSED]

Inputs: {email:user@example.com, password="User@123"}
Output: Access denied. Admin privileges required [PASSED]

Test Results:
returned token for valid credentials (1 ms)
rejected admin login for non-admin users (1 ms)

c. POST /api/v1/users/forgotPassword
This test ensures that the endpoint correctly initiates the password reset process. It verifies that a reset link is sent for valid users and that proper error messages are returned for unknown users.

Inputs: {email:test@example.com}
Output: Reset link sent to your mail [PASSED]

Inputs: {email:nonexistent@example.com}
Output: User Not Found [PASSED]

Test Results:
sent reset token for valid email (1 ms)
handled errors while sending email (19 ms)

d. PATCH /api/v1/users/resetPassword
This test ensures that the endpoint correctly handles password resetting. It verifies that the password is updated when a valid token is provided and that an error is returned if the token is invalid or expired.

Inputs: {token:abc123, password="NewPass@123"}
Output: Password Changed Successfully [PASSED]

Inputs: {token:expired123, password="NewPass@123"}
Output: User Not Found or Token Expired [PASSED]

Test Results:
updated password for valid reset token (3 ms)
rejected password update for expired/invalid token (2 ms)
validated token existence and expiration (1 ms)

e. POST /api/v1/users/updatePassword
This test ensures that the endpoint correctly handles updating a user's password. It verifies that the password is updated when the current password is valid and that an appropriate error is returned for invalid credentials.

Inputs: {currentPassword=”CurrPass@123”,
newPassword="NewPass@123",confirmNewPassword=”NewPass@123”}

Test Results:
updated password for valid current password (2 ms)
rejected password update for invalid current password (3 ms)
validated matching new and confirm passwords (2 ms)
Structural Coverage: Code Coverage: 100% and Branch Coverage: 100%

2. Cart Controller Unit Tests:

The test was conducted to verify the cart system functionality, ensuring that cart operations (get, add, update, remove, clear) work correctly under various conditions.

Unit Details: Functions: getCart, addToCart, updateCartItem, removeFromCart, clearCart
Test Owner: Saatvik Gundapaneni
Test Date: 03/04/25
Test Results: Passed. Under valid inputs, cart operations were performed successfully, while invalid inputs raised appropriate exceptions.

a. GET /api/v1/cart

This test ensures that the endpoint correctly retrieves a user's cart.

Inputs: ?username=testuser
Output: Cart object with populated items [PASSED]

Test Results:
- retrieved existing cart for valid username (2 ms)
- created new cart for first-time user (3 ms)
- rejected request with missing username (1 ms)

b. POST /api/v1/cart/add

This test verifies adding items to cart with proper validation.

Inputs:

```
{
   "username": "testuser",
   "productId": "65f7e8d12345",
   "quantity": 2
}
```

Output: Updated cart with new item [PASSED]

Test Results:
- added new item to cart successfully (3 ms)
- updated quantity for existing item (2 ms)
- validated product existence (2 ms)
- rejected invalid product ID (1 ms)
- rejected missing username or product ID (1 ms)

c. PATCH /api/v1/cart/update
This test ensures proper updating of cart item quantities.

Inputs:

```
{
   "username": "testuser",
   "productId": "65f7e8d12345",
   "quantity": 5
}
```

Output: Updated cart with new quantity [PASSED]

Test Results:
- updated item quantity successfully (2 ms)
- rejected quantity less than 1 (1 ms)
- handled non-existent cart item (2 ms)
- validated cart existence (1 ms)

d. POST /api/v1/cart/remove
This test verifies proper removal of items from cart.

Inputs:

```
{
   "username": "testuser",
   "productId": "65f7e8d12345"
}
```

Output: Updated cart with item removed [PASSED]

Test Results:
- removed item from cart successfully (2 ms)
- handled non-existent item removal (1 ms)

- validated required fields (1 ms)

e. POST /api/v1/cart/clear
This test ensures proper clearing of entire cart.

Inputs:

```
{
   "username": "testuser"
}
```

Output: Empty cart [PASSED]

Test Results:
- cleared cart successfully (2 ms)
- validated username requirement (1 ms)
- handled non-existent cart (1 ms)

Coverage Report:
- Function Coverage: 100%
- Statement Coverage: 100%
- Branch Coverage: 100%
- Line Coverage: 100%

All test cases passed successfully, verifying the robust functionality of the cart system under various scenarios including edge cases and error conditions. The system properly handles all CRUD operations while maintaining data integrity and proper validation.

3 .ProductController:
The test was to check Product Management components. How the component is reacting under right, wrong and empty inputs.
The tests conducted passed successfully.

Unit Details: Functions: getAllProducts, getProduct, createProduct, updateProduct, deleteProduct, updateAllProductsToOthers, getProductsBySeller, deleteSellerProduct, resizeProductImages
Test Owner: Voora Rakesh
Test Date: 03/04/2025

Test Results: Passed. Under right inputs, product operations were successful whereas under wrong and empty inputs appropriate exceptions were raised.

a. GET /api/v1/products

This test ensures that the endpoint correctly retrieves a list of all products from the database.
Inputs: {}
Output: Products Data: [{id: 1, name: "Product1"}, {id: 2, name: "Product2"}] [PASSED]
Test Results:
returned all products with seller populated (7 ms)
handled errors and send status 400 (1 ms)

b. GET /api/v1/products/:id

This test ensures that the endpoint correctly retrieves a specific product from the database based on id.
Inputs: {id=123}
Output: Product Data: {id: 123, name: "Product1", price: 100} [PASSED]
Inputs: {id=999}
Output: No product found with that ID [PASSED]
Test Results:
returned product if found (1 ms)
returned 404 if product not found (1 ms)
handled errors when fetching product (1 ms)

c. POST /api/v1/products

This test ensures that the endpoint correctly handles the creation of a new product. It verifies that valid product data is saved to the database and that appropriate responses are returned for both successful and failed operations.
Inputs: {
name="New Product",
price=100,
description="Test product",
images=[file1.jpg, file2.jpg]
}
Output: Product Data: {id: 456, name: "New Product", price: 100} [PASSED]
Inputs: {
name="",
price=-100
}
Output: Invalid product data [PASSED]
Test Results:
created a new product and returned 201 (2 ms)
returned error response for invalid product data (2 ms)
handled errors during product creation (2 ms)

d. PATCH /api/v1/products/:id

This test ensures that the endpoint correctly handles the updation of a product. It verifies that appropriate responses are returned for both successful and failed operations.

Inputs: {
  id=123,
  name="Updated Product",
  price=200
}
Output: Product Data: {id: 123, name: "Updated Product", price: 200} [PASSED]
Inputs: {
  id=999,
  name="Updated Product"
}
Output: No product found with that ID [PASSED]
Test Results:
    updated product if seller is the owner (2 ms)
    returned 404 if product not found (1 ms)
    returned 401 if user is not the seller (1 ms)

d. DELETE /api/v1/products/:id

This test ensures that the endpoint correctly handles the deletion of a product. It verifies that appropriate responses are returned for both successful and failed operations.

Inputs: {id=123}
Output: Product deleted successfully [PASSED]

Inputs: {id=999}
Output: No product found with that ID [PASSED]

Test Results:
    deleted product if seller or admin (1 ms)
    returned 404 if product not found
    returned 401 if not authorized to delete

Structural Coverage: Code Coverage: 100% and Branch Coverage: 100%

4. ProductReqController:

The test was to check Product Request Management components. How the component is reacting under right, wrong and empty inputs.
The tests conducted passed successfully.

Unit Details: Functions: getAllProductRequests, getUserProductRequests, createProductRequest, updateProductRequest, deleteProductRequest
Test Owner: Anirudh
Test Date: 03/04/2025
Test Results: Passed. Under right inputs, product request operations were successful whereas under
wrong and empty inputs appropriate exceptions were raised.

a. GET /api/v1/product-requests
This test ensures that the endpoint correctly retrieves all product requests in the system. It verifies that the returned data includes request IDs and descriptions and that the response structure is as expected.

Inputs: {}
Output: Product Requests Data: [{id: 1, description: "Request1"}, {id: 2, description: "Request2"}] [PASSED]

Test Results:
returned list of all product requests (2 ms)
validated correct structure of response data (1 ms)

b. GET /api/v1/product-requests/user-requests
This test ensures that the endpoint correctly retrieves product requests specific to a given user. It verifies that only requests associated with the provided username are returned.

Inputs: {username=testuser}
Output: User Requests Data: [{id: 1, description: "Request1", username: "testuser"}] [PASSED]

Test Results:
filtered product requests by username (2 ms)
returned only the user's own product requests (2 ms)

c. POST /api/v1/product-requests
This test ensures that the endpoint correctly creates a new product request. It verifies that valid inputs result in the creation of a request and that missing or invalid fields trigger appropriate validation errors.

Inputs: {
description="New Product Request",
category="Electronics"
}
Output: Product Request Data: {id: 456, description: "New Product Request", username: "testuser"} [PASSED]

Inputs: {

description=""
}
Output: Please Provide Product Description [PASSED]

Test Results:
successfully created new product request (3 ms)
validated required fields like description (2 ms)
returned error for missing description (2 ms)

d. PATCH /api/v1/product-requests

This test ensures that the endpoint correctly handles updates to product requests. It verifies that valid updates (such as description) are processed successfully, while restricted fields like username are not editable. It also ensures that users cannot modify product requests they do not own.

Inputs: {
id=123,
description="Updated Request"
}
Output: Product Request Data: {id: 123, description: "Updated Request"} [PASSED]

Inputs: {
id=123,
username="other user"
}
Output: Username cannot be updated [PASSED]

Inputs: {
id=123,
username="testuser"
}
Output: You cannot modify this request [PASSED]

Test Results:
successfully updated request description (3 ms)
blocked attempt to change username field (2 ms)
prevented unauthorized user from editing request (2 ms)

e. DELETE /api/v1/product-requests

This test ensures that the endpoint correctly handles the deletion of product requests. It verifies that requests are deleted when valid IDs are provided by authorized users and that appropriate errors are returned when unauthorized users attempt deletion.

Inputs: {id=123}
Output: Request Deleted [PASSED]

Inputs: {id=123, username="other user"}
Output: Not authorised to delete [PASSED]

Test Results:
deleted product request with valid ID and ownership (2 ms)
blocked deletion attempt by unauthorized user (2 ms)

Structural Coverage: Code Coverage: 100% and Branch Coverage: 100%

5.UserController:

The test was to check User Management components. How the component is reacting under right, wrong and empty inputs.
The tests conducted passed successfully.

Unit Details: Functions: getAllUsers, getUser, getUserForAdmin, updateUser, deleteUser, deleteUserByAdmin
Test Owner: Koushik
Test Date: 03/04/2025
Test Results: Passed. Under right inputs, user operations were successful whereas under wrong and empty inputs appropriate exceptions were raised.

a. GET /api/v1/users

This test ensures that the endpoint correctly retrieves a list of all users. It verifies that user data is returned in the expected format when no filters or parameters are applied.

Inputs: {}
Output: Users Data: [{id: 1, name: "User1"}, {id: 2, name: "User2"}] [PASSED]

Test Results:
returned full user list with expected structure (2 ms)
handled empty input without errors (1 ms)

b. GET /api/v1/users

This test ensures that the endpoint correctly returns data of the currently authenticated user. It verifies that the appropriate user object is returned with matching fields like id, name, and email.

Inputs: {}
Output: User Data: {id: 123, name: "User1", email: "user1@example.com"} [PASSED]

Test Results:
returned data for authenticated user (1 ms)
verified fields: id, name, and email (2 ms)

c. GET /api/v1/users/:id

This test ensures that the endpoint correctly fetches a specific user based on their ID. It verifies that the correct user details (such as name) are returned when a valid ID is provided.

Inputs: {id=123}
Output: User Name: "User1" [PASSED]

Test Results:
fetched user data for valid ID (2 ms)
validated returned name matches expected value (1 ms)

d. PATCH /api/v1/users

This test ensures that the endpoint correctly handles user profile updates. It verifies that valid updates (like name, username, photo) are processed successfully, while restricted fields like password are appropriately blocked.

Inputs: {
name="Updated Name",
username="new username",
photo=image.jpg
}
Output: User Data: {id: 123, name: "Updated Name", username: "new username"} [PASSED]

Inputs: {
password="newpass",
confirmPassword="newpass"
}
Output: Cannot Update Password Here [PASSED]

Inputs: {
username="existing user"
}
Output: Username already exists. Please choose a different username [PASSED]

Test Results:
successfully updated valid user fields (3 ms)
blocked password update via profile route (2 ms)
handled duplicate username with error message (3 ms)

e. DELETE /api/v1/users/:id

This test ensures that the endpoint correctly handles user account deletion. It verifies deletion both by confirming the user's password and by directly using their ID. Appropriate error messages are returned for invalid password attempts.

Inputs: {
currentPassword="correct pass"
}
Output: User deleted successfully [PASSED]

Inputs: {
currentPassword="wrong pass"
}
Output: Invalid password [PASSED]

Inputs: {id=123}
Output: User deleted successfully [PASSED]

Inputs: {}
Output: User ID: 123 [PASSED]

Test Results:

deleted user with valid password (2 ms)
rejected deletion with invalid password (2 ms)
deleted user via ID-based route (1 ms)
retrieved user ID from session for deletion context (1 ms)

Structural Coverage: Code Coverage: 100% and Branch Coverage: 100%

6. Order Controller Unit Tests:

The test was conducted to verify the order system functionality, ensuring that order operations (create, verify payment, get, update, cancel) work correctly under various conditions, with proper authentication.

Unit Details: Functions: createOrder, verifyPayment, getAllOrders, getOrder, getUserOrders, updateOrderStatus, cancelOrder
Test Owner: Sanjay
Test Date: 03/04/25
Test Results: Passed. Under valid inputs and proper authentication, order operations were performed successfully, while invalid inputs or unauthorized access were rejected.

a. POST /api/v1/orders (Protected Route)

This test ensures that the endpoint correctly creates new orders with Razorpay integration and proper authentication.

Inputs:

```
{
   "items": [
```

```
      {
         "product": "65f7e8d12345",
         "quantity": 2,
         "price": 1000
      }
   ],
   "totalAmount": 2000,
   "shippingAddress": "123 Test St, City"
}
```


Headers:


```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```


Output: Created order object with Razorpay details [PASSED]

Test Results:
- created new order with valid inputs and auth (4 ms)
- generated valid Razorpay order (3 ms)
- rejected unauthorized access (2 ms)
- rejected missing auth token (1 ms)
- rejected missing required fields (1 ms)
- validated total amount calculation (2 ms)

b. POST /api/v1/orders/verify-payment (Protected Route)
This test verifies payment verification with Razorpay and proper authentication.

Inputs:


```
{
   "razorpay_order_id": "order_123",
   "razorpay_payment_id": "pay_123",
   "razorpay_signature": "valid_signature"
}
```


Headers:


```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```


Output: Payment verification status [PASSED]

Test Results:
- verified valid payment signature with auth (3 ms)
- rejected unauthorized verification attempt (2 ms)
- rejected invalid signature (2 ms)

- updated order payment status (2 ms)
- cleared cart after successful payment (3 ms)
- handled missing payment details (1 ms)

c. GET /api/v1/orders
This test ensures proper retrieval of all orders (admin route).

Test Results:
- retrieved all orders successfully (2 ms)
- properly formatted response data (1 ms)
- handled empty orders list (1 ms)

d. GET /api/v1/orders/:id (Protected Route)
This test verifies fetching specific order details with authentication.

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Detailed order object [PASSED]

Test Results:
- retrieved order by valid ID with auth (2 ms)
- rejected unauthorized access (1 ms)
- handled non-existent order ID (1 ms)
- validated order ID format (1 ms)

e. GET /api/v1/orders/user/:username (Protected Route)
This test ensures proper retrieval of user-specific orders with authentication.

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Test Results:
- retrieved user orders successfully with auth (2 ms)
- rejected unauthorized access (1 ms)
- handled user with no orders (1 ms)
- validated username parameter (1 ms)

f. PATCH /api/v1/orders/:id (Protected Route)
This test verifies order status updates with authentication.

Inputs:

```
{
   "status": "delivered"
}
```

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Updated order status [PASSED]

Test Results:
- updated order status successfully with auth (2 ms)
- rejected unauthorized update attempt (1 ms)
- validated status values (1 ms)
- handled invalid order ID (1 ms)

g. PATCH /api/v1/orders/:id/cancel (Protected Route)
This test verifies order cancellation functionality with authentication.

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Test Results:
- cancelled valid order successfully with auth (2 ms)
- rejected unauthorized cancellation (2 ms)
- prevented cancelling delivered orders (1 ms)
- prevented duplicate cancellations (1 ms)
- validated user authorization (2 ms)

Coverage Report:
- Function Coverage: 100%
- Statement Coverage: 100%
- Branch Coverage: 100%
- Line Coverage: 100%

All test cases passed successfully, verifying the robust functionality of the order system under various scenarios including authentication, payment processing, status updates, and error conditions. The system properly handles all operations while maintaining data integrity, proper validation, and security through authentication middleware.

7 .ChatController:

The test was to check Chat Management components. How the component is reacting under right, wrong and empty inputs.
The tests conducted passed successfully.

Unit Details: Functions:createChat, getUserChats, getChatMessages, saveMessage, createOrFindChatWithUser
Test Owner: Rakesh
Test Date: 06/04/2025
Test Results: Passed. Under right inputs, chat operations were successful whereas under wrong and empty inputs appropriate exceptions were raised.

a. POST /api/v1/chats (createChat)

Inputs:

```
{
 "userId": "67ee74b13407c16157504e3c",
 "participantId": "67ee736972a7e08a66cd183f"
}
```

Headers:
{ "Content-Type" : "application/json"
"Authorization": "Bearer <valid_jwt_token>"
}

Outputs: New chat created with 201 status(Passed).

Test Results:
- created a new chat with valid inputs(730 ms)
- handled invalid inputs(Only one user or Invalid userId) and got 400 Bad Request

b.GET /api/v1/chats/user/:userId (getUserChats)

Inputs:

```
{
 "userId": "67ee736972a7e08a66cd183f"
}
```

Headers:
{ "Content-Type" : "application/json"
"Authorization": "Bearer <valid_jwt_token>"
}

Output : Successfully retrieved 4 chats for user "vrakesh23" with populated participant details and unread counts.

Test Results:
- successfully retrieved all chats for a valid user ID (4 chats found) (478 ms)
- Maintained correct sort order (newest chats first) (24 ms)
- Accurately calculated unread message counts (all 0) (2 ms)
- handled the case of invalid user Id and got status 500 Internal Server Error

c. **GET /api/v1/chat/:chatId/messages**

Inputs:

GET 127.0.0.1:3000/api/v1/chat/67ee7ca3317e30090011861c/messages

Headers:
{ "Content-Type" : "application/json"
"Authorization": "Bearer <valid_jwt_token>"
}

Output: Successfully retrieved 9 messages for chat ID 67ee7ca3317e30090011861c(status 200 OK)

Test Results:
- Returned all messages for valid chat ID (9 messages)(134 ms)
- Correctly populated sender/receiver details(3 ms)
- Included full message metadata (content, timestamps)(5 ms)
- handled the case of invalid chat Id and got empty chat(92 ms)

d. **POST /api/v1/chat/message**

Inputs:

```
{
  "chatId": "65f123abc456def789abcdef",
  "senderId": "65e123abc456def789abcdef",
  "receiverId": "65e123abc456def789abcfed",
  "content": "Hello, how are you?"
}
```

Headers:
{ "Content-Type" : "application/json"
"Authorization": "Bearer <valid_jwt_token>"
}

Outputs:Successfully created new message in chat 65f123abc456def789abcdef (ID:

67f2a0ba323c4827da3ff4cb)(Status : 201 Created)

Test Results:

- Created a new message with valid inputs (163 ms)

- Correctly assigned chat/sender/receiver IDs (3 ms)
- Returned 201 status for successful creation (2 ms)
- handled the case of invalid inputs and got 500 Internal Server Error(52 ms)

e. **POST /api/v1/chat/with-user/:userId**

Inputs:

POST 127.0.0.1:3000/api/v1/chat/with-user/67ee74c7317e3009001184f7

Headers:
{ "Content-Type" : "application/json"
"Authorization": "Bearer <valid_jwt_token>"
}

Output: Successfully retrieved existing chat between **vrakesh23** and **sanjayra23** (ID: 67ee7ca3317e30090011861c) with last message *"Hello"*.(Status : 200 OK)

Test Results:
- Found and returned existing chat between valid users (139 ms)
- Correctly populated participant usernames (3 ms)
- Included last message metadata with read status (3 ms)
- handled the case of invalid userId and got 500 Internal Server Error

Coverage Report:
- Function Coverage: 100%
- Statement Coverage: 100%
- Branch Coverage: 100%
- Line Coverage: 100%

All test cases passed successfully, verifying the robust functionality of the messaging system under various scenarios including edge cases and error conditions.

8. Auction controller:
The test was conducted to verify the auction system's functionality, ensuring that creating, bidding, and ending auctions work correctly under various conditions.

Unit Details: Functions: createAuction, getActiveAuctions, getAuction, placeBid, endAuction
Test Owner: saatvik
Test Date: 05/04/25
Test Results: Passed. Under valid inputs, auctions were created and managed successfully, while invalid inputs raised appropriate exceptions.

a. POST /api/v1/auctions
This test ensures the endpoint correctly handles auction creation with proper validation.

Inputs:
{

"productId": "65f7e8d12345",
"startingPrice": 1000,
"startTime": "2024-03-20T10:00:00Z",
"endTime": "2024-03-21T10:00:00Z",
"seller": "65f7e8d12346",
"bidIncrement": 10
}

Output: Created auction object with status 'active' [PASSED]

Test Results:
- created a new auction with valid inputs (3 ms)
- rejected auction creation with missing required fields (2 ms)
- validated product and seller existence (4 ms)

b. GET /api/v1/auctions
This test verifies the retrieval of active auctions with proper population of related data.

Test Results:
- returned list of active auctions (2 ms)
- properly populated product and seller information (3 ms)
- filtered out non-active auctions (1 ms)

c. GET /api/v1/auctions/:id
This test ensures proper retrieval of single auction details.

Inputs: Valid auction ID
Output: Detailed auction object with populated references [PASSED]

Test Results:
- returned auction details for valid ID (2 ms)
- handled invalid auction ID format (1 ms)
- handled non-existent auction ID (1 ms)

d. POST /api/v1/auctions/:id/bid
This test verifies the bidding functionality with proper validation.

Inputs:

{
"bidAmount": 1100,
"bidderId": "65f7e8d12347",
"bidderName": "Test Bidder"
}

Output: Updated auction with new bid [PASSED]

Test Results:
- accepted valid bid above minimum increment (2 ms)

- rejected bid below minimum increment (1 ms)
- prevented seller from bidding on own auction (1 ms)
- rejected bids on ended auctions (1 ms)
- validated bidder existence (2 ms)

e. PATCH /api/v1/auctions/:id/end
This test ensures proper auction ending functionality.

Test Results:
- successfully ended active auction (3 ms)
- properly set winner and final price (2 ms)
- rejected ending already ended auctions (1 ms)
- handled auction not found scenario (1 ms)

Coverage Report:
- Function Coverage: 100%
- Statement Coverage: 100%
- Branch Coverage: 100%
- Line Coverage: 100%

All test cases passed successfully, verifying the robust functionality of the auction system under various scenarios including edge cases and error conditions.

9 WishlistController:
The test was conducted to verify the wishlist system functionality, ensuring that wishlist operations (check, get, add, remove, clear) work correctly under various conditions, with proper authentication.

Unit Details: Functions: checkWishlistItem, getWishlist, addToWishlist, removeFromWishlist, clearWishlist
Test Owner: Saatvik
Test Date: 03/04/25
Test Results: Passed. Under valid inputs and proper authentication, wishlist operations were performed successfully, while invalid inputs or unauthorized access were rejected.

a. GET /api/v1/wishlist/check/:productId (Protected Route)
This test ensures that the endpoint correctly checks if an item exists in user's wishlist.

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Wishlist item status [PASSED]

Test Results:
- checked item existence with valid auth (2 ms)

- returned false for non-existent wishlist (1 ms)
- rejected unauthorized access (1 ms)
- validated product ID format (1 ms)

b. GET /api/v1/wishlist (Protected Route)
This test ensures proper retrieval of user's wishlist.

Query Parameters: ?username=testuser
Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: User's wishlist data [PASSED]

Test Results:
- retrieved existing wishlist with auth (2 ms)
- created new wishlist for first-time user (3 ms)
- rejected unauthorized access (1 ms)
- rejected missing username (1 ms)

c. POST /api/v1/wishlist (Protected Route)
This test verifies adding items to wishlist.

Inputs:

```
{
   "username": "testuser",
   "productId": "65f7e8d12345"
}
```

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Updated wishlist with new item [PASSED]

Test Results:
- added new item successfully with auth (3 ms)
- prevented duplicate items (2 ms)
- validated product existence (2 ms)
- rejected unauthorized access (1 ms)
- rejected invalid product ID (1 ms)
- rejected missing required fields (1 ms)

d. DELETE /api/v1/wishlist/remove (Protected Route)

This test verifies removal of items from wishlist.

Inputs:

```
{
   "username": "testuser",
   "productId": "65f7e8d12345"
}
```

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Updated wishlist with item removed [PASSED]

Test Results:
- removed item successfully with auth (2 ms)
- handled non-existent item removal (1 ms)
- rejected unauthorized access (1 ms)
- validated required fields (1 ms)

e. DELETE /api/v1/wishlist/clear (Protected Route)
This test ensures proper clearing of entire wishlist.

Inputs:

```
{
   "username": "testuser"
}
```

Headers:

```
{
   "Authorization": "Bearer <valid_jwt_token>"
}
```

Output: Empty wishlist [PASSED]

Test Results:
- cleared wishlist successfully with auth (2 ms)
- rejected unauthorized access (1 ms)
- validated username requirement (1 ms)
- handled non-existent wishlist (1 ms)

Coverage Report:

- Function Coverage: 100%
- Statement Coverage: 100%
- Branch Coverage: 100%
- Line Coverage: 100%

All test cases passed successfully, verifying the robust functionality of the wishlist system under various scenarios including authentication, item management, and error conditions. The system properly handles all operations while maintaining data integrity, proper validation, and security through authentication middleware.

3 Integration Testing

1. Login Process

Module Details: Login
Test Owner: Ayush Yadav
Test Date: 03/04/2025-03/04/2025
Test Results:

- Frontend (Login.jsx) successfully integrates with backend (authController.js)
- JWT token generation and storage works correctly
- Session management across components is maintained
- Error handling for invalid credentials is consistent
- Redirection to appropriate pages based on user role works as expected

2. Registration Process

Module Details: SignUp
Test Owner: Rakesh
Test Date: 03/04/2025-03/04/2025
Test Results:

- Email validation between frontend and backend is synchronized
- OTP generation and verification process works end-to-end
- User data validation is consistent across layers
- Error messages are properly propagated from backend to frontend
- Profile creation and initial data setup is successful

3 Password Reset

Module Details: ForgotPassword
Test Owner: Koushik
Test Date: 03/04/2025-03/04/2025
Test Results:

- Password reset flow maintains security constraints
- Session handling during password reset is secure
- Error states are properly handled and displayed

4. Product Listing
Module Details: ProductDisplay

Test Owner: Sanjay
Test Date: 03/04/2025-03/04/2025
Test Results:

- Product data fetching from backend to frontend works correctly
- Image loading and display is optimized
- Filtering works as expected
- Real-time updates are synchronized
- Error handling for failed requests is implemented

5. Product Request

Module Details: ProductRequest
Test Owner: Anirudh
Test Date: 03/04/2025-03/04/2025
Test Results:

- Admin approval workflow functions correctly
- Error handling for invalid requests

6. Cart Operations

Module Details: CartManagement
Test Owner: Saatvik
Test Date: 03/04/2025-03/04/2025
Test Results:

- Cart state management is synchronized
- Real-time updates work correctly
- Price calculations are accurate

7. Order Processing

Module Details: OrderProcessing
Test Owner: Yashwanth
Test Date: 03/04/2025-03/04/2025
Test Results:

- Order creation and validation works end-to-end
- Payment integration is secure and reliable
- Order status updates are real-time
- Email notifications are properly triggered
- Error handling for failed transactions

8. Profile Update

Module Details: ProfileManagement
Test Owner: Sanjay
Test Date: 03/04/2025-03/04/2025
Test Results:

- Profile data updates are synchronized
- Image upload and processing works correctly
- Validation rules are consistently applied
- Real-time updates are reflected
- Error handling for invalid inputs

9. Direct Messaging

Module Details: ChatSystem
Test Owner: Prem
Test Date: 03/04/2025-03/04/2025
Test Results:

- Real-time messaging works correctly
- Message delivery is reliable
- Error handling for failed messages

10. Auction Management

Module Details: AuctionSystem
Test Owner: Pranaya
Test Date: 03/04/2025-03/04/2025
Test Results:

- Auction creation and setup works correctly
- Bidding system is synchronized
- Real-time updates for bids
- Auction end process functions properly

11. Bidding System

Module Details: BiddingSystem
Test Owner: Rakesh
Test Date: 03/04/2025-03/04/2025
Test Results:

- Bid placement and validation works
- Price updates are real-time
- Winner determination is accurate
- Payment processing is integrated
- Error handling for bidding failures

12. Wishlist/Favourites Integration

Module Details: FavouritesModule
Test Owner: Anirudh
Test Date: 03/04/2025-04/04/2025
Test Results:

- Users can add/remove products from favourites
- State sync between backend and frontend is smooth
- Proper error handling on invalid item operations

13. Sell History

Module Details: SellHistory
Test Owner: Yashwanth
Test Date: 03/04/2025-04/04/2025
Test Results:

- Sellers can view all their listed products (regular & auction)
- Proper separation between auction-type and direct-sale listings
- Backend fetches accurate data based on seller ID

14. Search and Filtering

Module Details: SearchFilter
Test Owner: Meghana
Test Date: 03/04/2025-04/04/25
Test Results:

- Keyword-based search returns accurate product names
- Category-based filtering works as expected
- Min and Max price filtering correctly narrows results
- Error handling for invalid inputs (e.g., negative price)

Additional Comments:

- All tests were performed in a development environment
- Test data was properly isolated and cleaned up after each test
- Security checks were implemented for sensitive operations
- Error handling was verified across all integration points

- Real-time functionality was tested under various network conditions
- Cross-browser compatibility was verified

4 System Testing

1. Requirement: Chat with other users(Between Buyer and Seller)

Users will be able to chat with other users.

Test Owner: [Rakesh, Koushik]
Test Date: [01/04/2025]-[03/04/2025]
Test Results: [Success]

During the testing phase conducted from [01/04/2025] to [03/04/2025], we rigorously evaluated the functionality of the chat feature, adhering to the specified requirements through a combination of manual and automated testing.

● Manual Testing: We initiated new chat creation, sent messages, and verified whether the receiver was getting the messages in proper latency or not.
● Automated Testing: Testing tools such as Selenium were utilized to simulate user interactions and verify the proper functioning of the chat creation and messages received in proper latency.

Result: The chat feature effectively implemented the specified requirements, allowing users to initiate and engage in conversations with other users while supporting the exchange of text messages. Both manual and automated testing methodologies confirmed the robustness of the feature.

2. Requirement: User Authentication

User Registration :

All users must register on the system using their IITK Email ID, and a verification mail is sent to their email ID.

Changing Password for Log In :

If a user forgets his/her password for logging in, the user will be able to reset the login password via mail sent to the registered email id.

Test Owner: [Anirudh]
Test Date: [01/04/2025]-[03/04/2025]
Test Results: [Success]

The test was successfully conducted from [01/04/2025] to [03/04/2025], encompassing the updated requirements for user authentication. For user registration, the process now involves users registering using their IITK email ID and receiving a mail for verification. Additionally, the functionality to reset the login password via mail sent to the registered email ID was tested successfully.

During testing, various login scenarios were covered:

Successful login with the correct email and password.
Unsuccessful login with correct email but wrong password.
Unsuccessful login with incorrect email.

Similarly, for registration, the following cases were addressed:

Attempting to register with an already registered email.
Entering an invalid email address.

Result: All test cases yielded positive results, affirming the effectiveness of the user authentication system.

---

3. Requirement: Product and Auction Listing & Management

Users should be able to add and delete product and auction listings.

Test Owner: [Rakesh]

Test Date: [01/04/2025]-[03/04/2025]

Test Results: [Success]

The product management system was evaluated for accuracy, UI consistency, and database integrity.

- **Manual Testing:** Products were added and deleted to ensure updates were reflected instantly. Image uploads were also tested.

- **Automated Testing:** Bulk product uploads and deletions were simulated to test performance under high data load. Database integrity was checked after multiple operations.

Result:
Product and Auction listing and management worked efficiently, allowing users to manage their inventory seamlessly without issues.

---

4. Requirement: Product Search & Filtering

Users should be able to search for products and apply filters.

- **Test Owner:**[Sanjay]
- **Test Date:**[01/04/2025]-[03/04/2025]
- **Test Results:**[Success]

Product Search and filtering functionalities were evaluated for speed, accuracy, and relevance.

- **Manual Testing:** Searches were performed using different keywords, and filters such as price, category, and sorting were applied. The accuracy of results was verified.
- **Automated Testing:** Large-scale indexing and performance tests were conducted with thousands of products to measure search efficiency.

Result:
The search system delivered fast and precise results, enhancing product discoverability.

---

5. Requirement: Bidding & Auctions

Users should be able to place and track bids in real-time.

- **Test Owner:** [Prem, Meghana]
- **Test Date:** [01/04/2025] - [03/04/2025]
- **Test Results:** [Success]

The bidding system was validated to ensure accurate and real-time auction handling.

- **Manual Testing:** Multiple users placed bids, and the highest bid was verified to update instantly after the auction expired. Bid expiry and mails to both seller and bid winner were tested.

- **Automated Testing:** Simulated auctions with many bidders to assess system stability and fairness in bid processing.

Result:
The auction system handled competitive bidding accurately, ensuring fair transactions in real time.

---

6. Requirement: Test Payment Gateway Integration

Users should be able to complete test payments securely.

- **Test Owner:**[Yashwanth]

- **Test Date:** [01/04/2025] - [03/04/2025]

- **Test Results:** [Success]

Payment functionality was tested for transaction accuracy, security, and error handling.

- **Manual Testing:** Payments were processed using different methods, and failed transactions were validated for correct error messages.

- **Automated Testing:** High-volume transactions were simulated to test gateway reliability. Encryption security and compliance were also reviewed.

Result:
The payment system functioned securely, ensuring smooth and protected transactions.

---

7. Requirement: Order History & Transaction Records

Users should be able to view past orders and transaction details.

- **Test Owner:** [Sanjay]

- **Test Date:** [03/04/2025] - [05/04/2025]

- **Test Results:** Success

The order history system was evaluated for accuracy and fast retrieval of past transactions.

- **Manual Testing:** Various purchases were made, and transaction details, and order statuses were reviewed.
- **Automated Testing:** Thousands of transactions were simulated to test database indexing and retrieval speed.

Result:
Users could reliably access their complete transaction history with accurate details.

8. Requirement: Wishlist (Favourites Feature)

Users should be able to add and remove products from their wishlist.

Test Owner: [Saatvik]

Test Date: [01/04/2025] - [03/04/2025]

Test Results: [Success]

The wishlist feature was tested to ensure it worked correctly, saved data properly, and was easy to use.

- **Manual Testing:** Products were added and removed from the wishlist, and changes were verified to stay even after refreshing or logging out.

- **Automated Testing:** Multiple users updated their wishlists at the same time, and the system was checked to ensure data remained accurate after a server restart.

Result:
The wishlist feature functioned smoothly, allowing users to save and track their favorite products without issues.

9. Requirement: Testing Non-Functional Requirements – Software Quality Attributes

- **Usability:** The platform is designed with a user-friendly interface, ensuring an intuitive experience for buyers and sellers. Users can easily list products, place bids, manage orders, and engage in chat conversations without requiring prior technical knowledge.

The navigation flow is simple, with clear action buttons and an organized layout to enhance accessibility.

- **Performance & Scalability:** The platform is optimized to handle multiple concurrent users without lag or downtime. It supports real-time bidding, instant messaging, and fast product searches even under high-traffic conditions. Load tests have been conducted to verify performance under peak usage scenarios.

- **Security:** The application ensures robust security measures, including encrypted user authentication, secure test payment transactions. Sensitive data such as passwords are hashed, and user sessions are managed securely to prevent breaches.

- **Portability:** The platform is built to function smoothly across different devices, including desktops, tablets, and smartphones. The responsive design ensures a consistent experience, adapting seamlessly to varying screen sizes without compromising usability.

- **Interoperability:** The system is designed to integrate efficiently with third-party services such as payment gateways, and cloud storage. Users can switch between devices and resume their activities without losing data, ensuring a continuous and smooth user experience.

Test Owner: [Yashwanth, Saatvik]
Test Date: [03/04/2025]-[05/04/2025]
Test Results: [Success]

During the testing phase from 03/04/2025 to 05/04/2025, we assessed the platform's usability, performance, security, portability, and interoperability to ensure compliance with non-functional requirements.

- **Usability:** Users found the platform intuitive, with straightforward navigation and well-structured workflows. Testing confirmed that all key actions, including product listing, bidding, and messaging, were easily accessible.

- **Performance & Scalability:** The system was tested under heavy traffic and handled multiple users, transactions, and chats smoothly without slowing down.

- **Security:** Login and access controls were checked for weaknesses. User data, including passwords and payments, was securely encrypted and protected.

- **Portability:** The application functioned consistently across multiple devices and browsers. UI responsiveness and performance remained optimal regardless of screen

size.

- **Interoperability:** The platform connected seamlessly with external services like payment systems and real-time notifications in the messaging system, ensuring smooth transactions and data updates.

Result:

The application met the non-functional requirements, delivering a secure, high-performing, and user-friendly experience across different devices and environments.

5 Conclusion

How Effective and exhaustive was the testing?

Testing was extremely comprehensive, testing all important features with integration and unit tests. Unit tests tested each part of the code and had 100% coverage, and integration tests verified that features such as login, messaging, and transactions were working. Stress testing validated that the system was stable even with a lot of users at once. Both manual and automated testing assisted in verifying data accuracy between various user roles. Selenium and Postman tools ensured the process was smooth and efficient.

Which components have not been tested adequately?

Some of the components that involved mocking have not been tested using an automation suite. However, it was tested thoroughly manually with the help of some extensions.

What difficulties have you faced during testing?

The manual testing was time-consuming, but we tried to utilize the automated testing process as much as possible, which helped us complete it in less time. Therefore, we did not encounter any major difficulties.

How could the testing process be improved?

The testing process can be enhanced by integrating CI/CD pipelines for automated test execution on every code change. Increasing test coverage for edge cases and expanding automated regression tests would help detect issues early. Additionally, refining test cases periodically and optimizing test execution speed will improve efficiency and reliability.

Appendix A - Group Log

S.No	Date	Activity	Location	Members
1	30-03-2025	We organized meetings to initiate the testing procedures and in case of any issue.	RM	All
2	31-03-2025	Discussed and distributed work related to the issues in the backend testing and gave this works on the frontend. people.	RM	All
3	01-04-2025	We discussed and allocated tasks related to frontend issues and integration testing. Subsequently, we assigned the tasks to the backend team, who were responsible for developing the backend.	RM	All
4	03-04-2025	This was a 4-hour session during which we tested all the backend components using the Postman tool. We then assigned additional work to the frontend team.	RM	All
5	04-04-2025	This was a 3-hour session during which we tested the search and chat functionalities using automated tools. Additionally, we evaluated the non-functional requirements of the project and submitted the user manual.	RM	All
6	05-04-2025	We distributed the work to all team members for writing their respective parts in the testing document.	RM	All
7	06-04-2025	A final meeting was held to review our testing document.	RM	All