Tutorial 5.5: Testing

Objective

In this tutorial students shall be exposed to different types of test planning and shall implement the tests. This is a continuation of Tutorial 4: Testing

Integration Testing

Integration tests verify that the connections between our application code and any other part of the system (3rd party api, database) work as expected.

If your tests affect database data then it should be an integration test. When you have tests that affect database data.

Integration tests can also validate several business rules of the system.

Test	Dependencies	Description
test_create_user()	create_user() get_user()	Ensure user record in database has the correct values
test_authenticate()	create_user() authenticate()	Ensures authenticate() returns true when given the correct credentials
test_get_all_users_json()	create_user() get_all_users_json()	Verifies json data of all users in the system
test_update_user()	create_user() update_user() get_user()	Ensures an updated user is retrieved with the updated values
test_staff_create_listing()	create_staff() create_user() create_game() staff.create_listing()	Lets a staff create a game listing for a game a customer owns. Ensures the listing record is created with the corresponding staff, customer and game IDs.
test_staff_confirm_rental()	get_listing() get_user() create_rental_payment() staff.confirm_rental()	Ensures a rental and payment records are created with the appropriate IDs and parameters. The corresponding listing record should be updated to “unavailable”
test_staff_return_rental()	get_listing() get_rental() update_rental() create_payment() staff.confirm_return()	Ensures the return date of the corresponding rental record is updated, the listing should be updated to “available” and a payment record should be created if the return is late

The tests here are high level and describe what other controllers or methods are required to fulfil a given feature.

API Testing

API Tests are automated tests that ensure our API confirms to the specification. Your API Spec also serves as a test plan.

Description	Method	URL	Request Headers, Body	Response Body, Status

Create account	POST	/signup	{
			“username”:<string>,
			“password”:<string>
			}
				Success
				{
				“message”:”account created”
				}, 201
				
				Bad Username
				{
				“error”: “username already taken”
				}, 400

Login	POST	/auth	{
			“username”:<string>,
			“password”:<string>
			}
				Success
				{
				“token”:<token>
				}, 200
				
				Bad Credentials
				{
				“error”: “invalid credentials”
				}, 400

(Customer) Enlist Game	POST	/listings	Authorization: JWT <token>
			
			{
			“gameid”:<id>,
			“owner”:<id>,
			“condition”:< ‘good’ | ‘fair’ | ‘bad’ >,
			“price”: <float>
			}
				Success
				{
				“message”:”listing created”
				}, 201
				
				When token not provided
				{
				“error”:”not authenticated”
				}, 401
				
				Bad Game id
				{
				“error”:”game id <gameid> not found”
				}, 404
				
				Bad Condition
				{
				“error”:”<condition> not a valid condition”
				}, 400

Show game	GET	/listings	?platform=< NSW	[

listings			|PS5|XBOX|PC>	{
				“listingId”: <id>,
				“ownerId”: <id>,
				“gameId”: <id>,
				“condition”: < ‘good’ | ‘fair’ | ‘bad’ >,
				“status”: ‘listed’,
				“price”: <float>,
				“game”: {
				“gameId”:<id>,
				“title”:<string>,
				“rating”: <string>,
				“platform”: <platform>,
				“boxart”: <url>,
				“genre”: <string>
				}
				}
				]

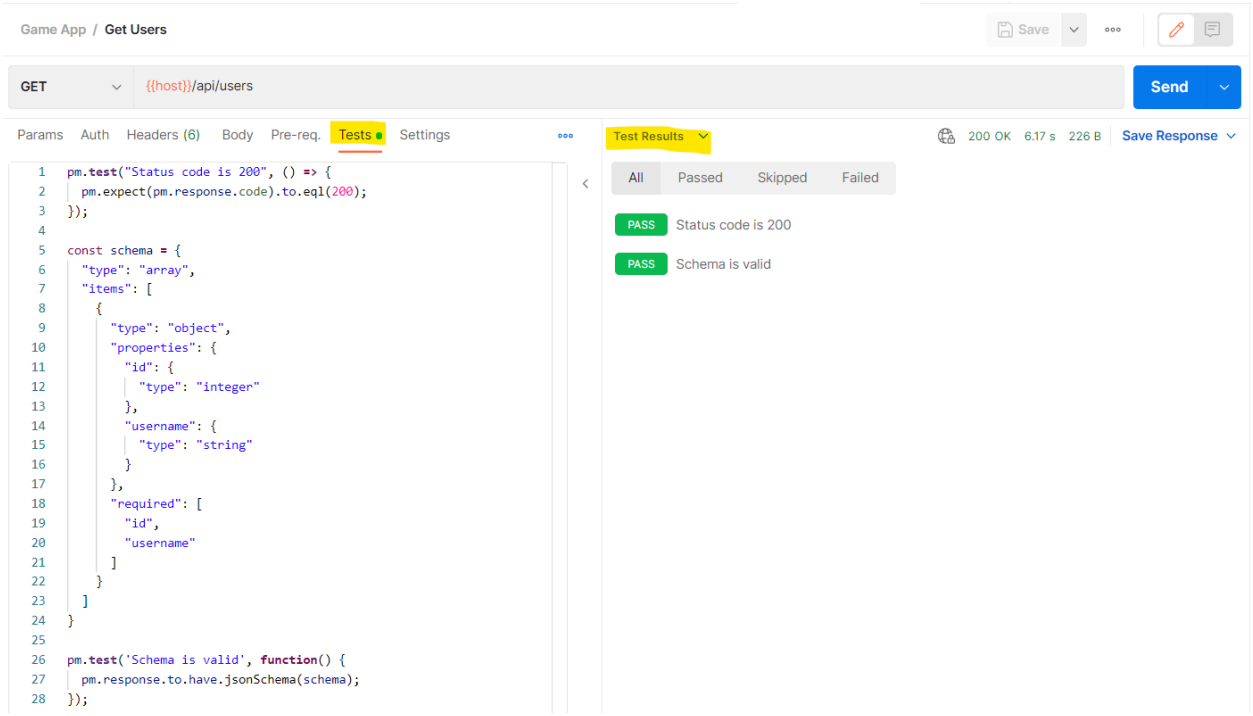
(Staff)
Make Payment
	POST	/payment	Authorization: JWT <token>
			
			{
			“amount” :<float>
			}
				{
				“message”: “payment created”,
				“paymentId”: <id>
				}, 201

(Staff)
Borrow Game
	POST	/rentals	Authorization: JWT <token>
			
			{
			“listingId”: <id>
			“customerId” :<id>
			}
				Success
				{
				“message”: “rental created”
				}, 201
				Bad id
				{
				“error”: “Bad listing id given”
				}, 404

(Staff)
Return Game
	UPDATE	/rentals/<rentalId>	Authorization: JWT <token>
			{
			“payment”:{
			“amount”: “float”
			}
			} or
			{
			“palymentId: <id>
			}
				Success
				{
				“message”: “rental updated”
				}, 201

(Customer)
Confirm Sell Game
	UPDATE	/listing/<listingId>	Authorization: JWT <token>
			
			{
			“status”: “sold”
			}
				Success
				{
				“message”: “listing removed from customer”
				}

API Tests are made in [postman scripts](#).



View the full [Postman Collection](#) (new link)

Question 1: Implement the integration tests from question 2 in a forked version of the [gamerental backend repo](#)

Question 2: Implement the route specified in question 3

Question 3: Implement a postman script to test the route specified in question 3