Tutorial 4: Testing

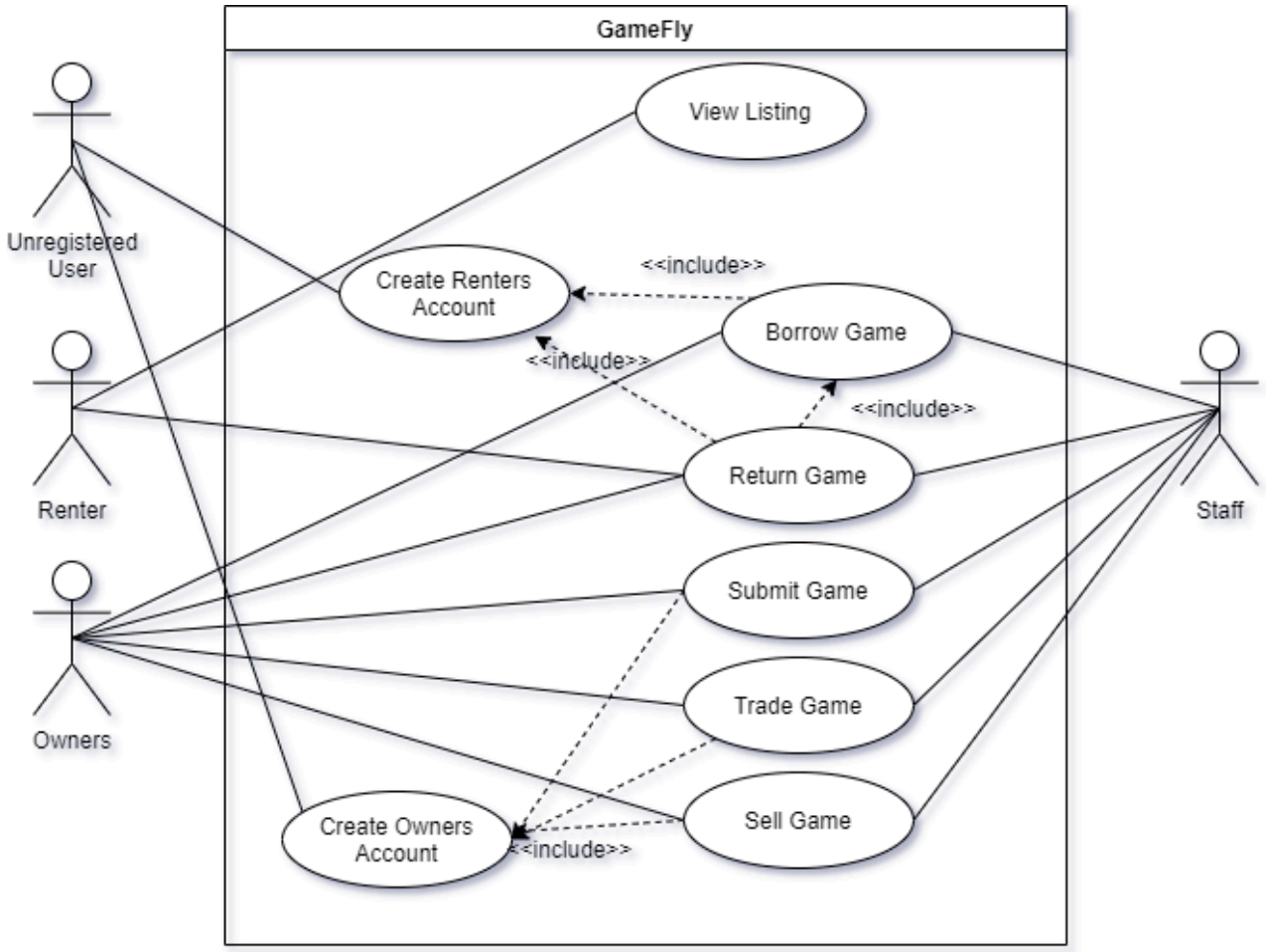
Objective

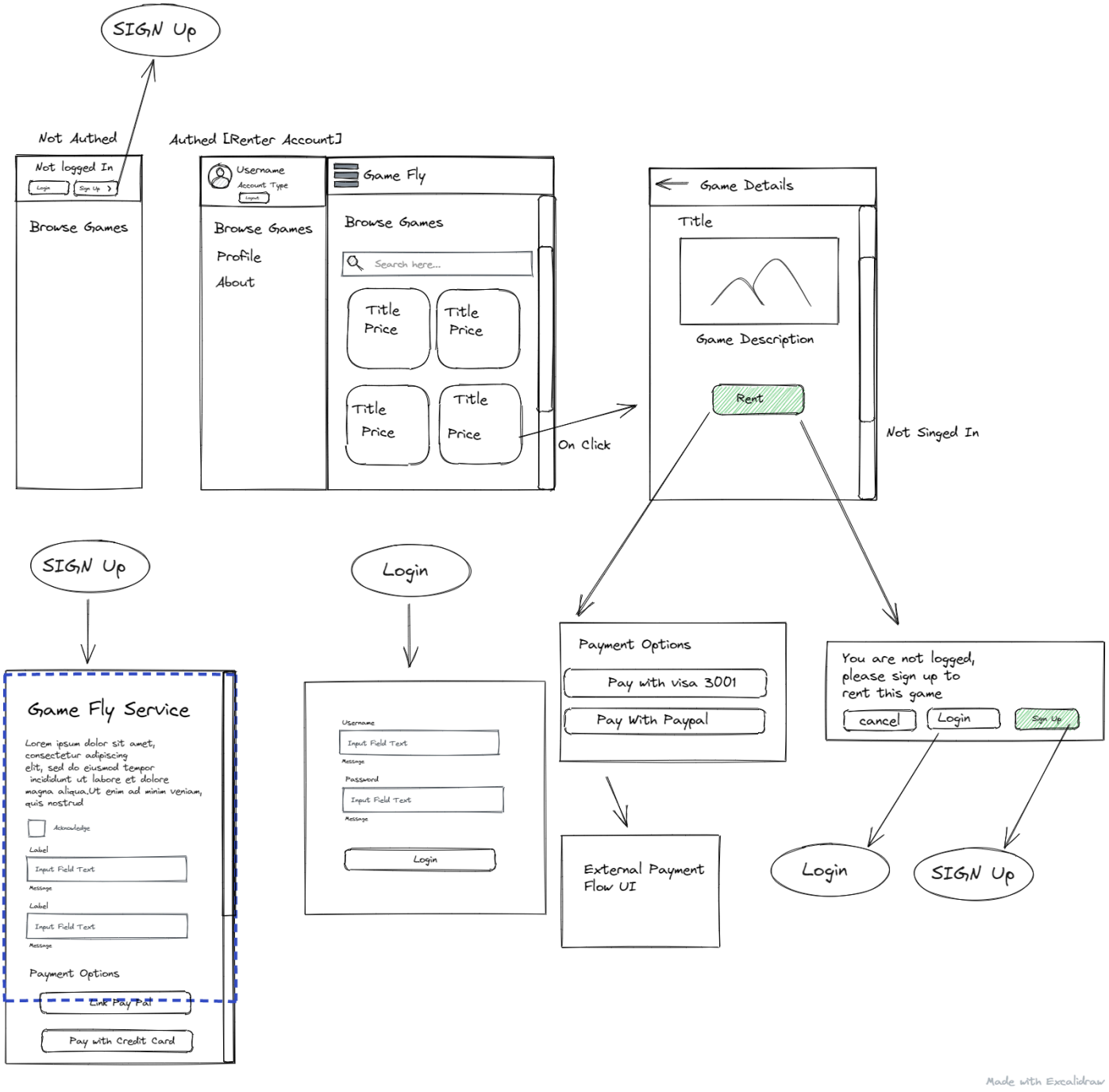
In this tutorial students shall be exposed to different types of test planning and shall implement the tests.

User Acceptance Testing

User acceptance tests help engineers verify if the software works **as the user intended**. In test planning, a User Acceptance Test report is drafted based on the requirements collected by the user and the Use Cases and UI designs created to meet those requirements.

The following use case and wireframe excerpt:





Made with Excalidraw

Can have the following test plan.

Test Case	Pre-conditions	Test Steps	Test Criteria	Success
Test Account Creation	None	☐ Expand nav bar, click on sign up ☐ A signup modal form appears ☐ Fill out the form with valid data ☐ Press sign up button	☐ User is alerted that signup is successful ☐ Navbar is changed to the authed state showing the user name and profile ☐ Logout button is shown in place of the signup/login buttons	
Test Login	None	☐ Expand nav bar, click on login ☐ A login modal form appears ☐ Fill out the form with valid data ☐ Press login button	☐ User is alerted that signup is successful ☐ Navbar is changed to the authed state showing the user name and profile ☐ Logout button is shown in place of the signup/login buttons	
Test Listing Game	☐ Must be logged on as a customer	☐ Navigate to My Listings page ☐ Click on add new listing ☐ Complete listing form ☐ Press add listing	☐ User is alerted that listed is created ☐ The new listing should appear on the My Listings page	
Test Browsing Game Listings	☐ None	☐ Scroll through the game listings ☐ Click on a desired listing	☐ Users should see expanded details on the clicked listing.	
Test game rental	☐ A game listing should exist and be available ☐ Customer should be logged in	☐ Select game from catalogue ☐ Click on rent on game details screen ☐ Finalise payment on payment integration	☐ Payment successful prompt ☐ Redirect to rentals view ☐ Game should appear on user rentals view	
Test game rental not logged in	☐ None	☐ Select game from catalogue ☐ Click on rent button on game details screen	☐ Customer user should have an outstanding rental	
Test Game Return	☐ Customer user should have an outstanding rental	☐ User returns physical game to staff ☐ Users pays any late fees	☐ Customer user should see their transaction appear in their transaction history ☐ Customer user should be able to rent other games	
Test Sell Game	☐	☐	☐	

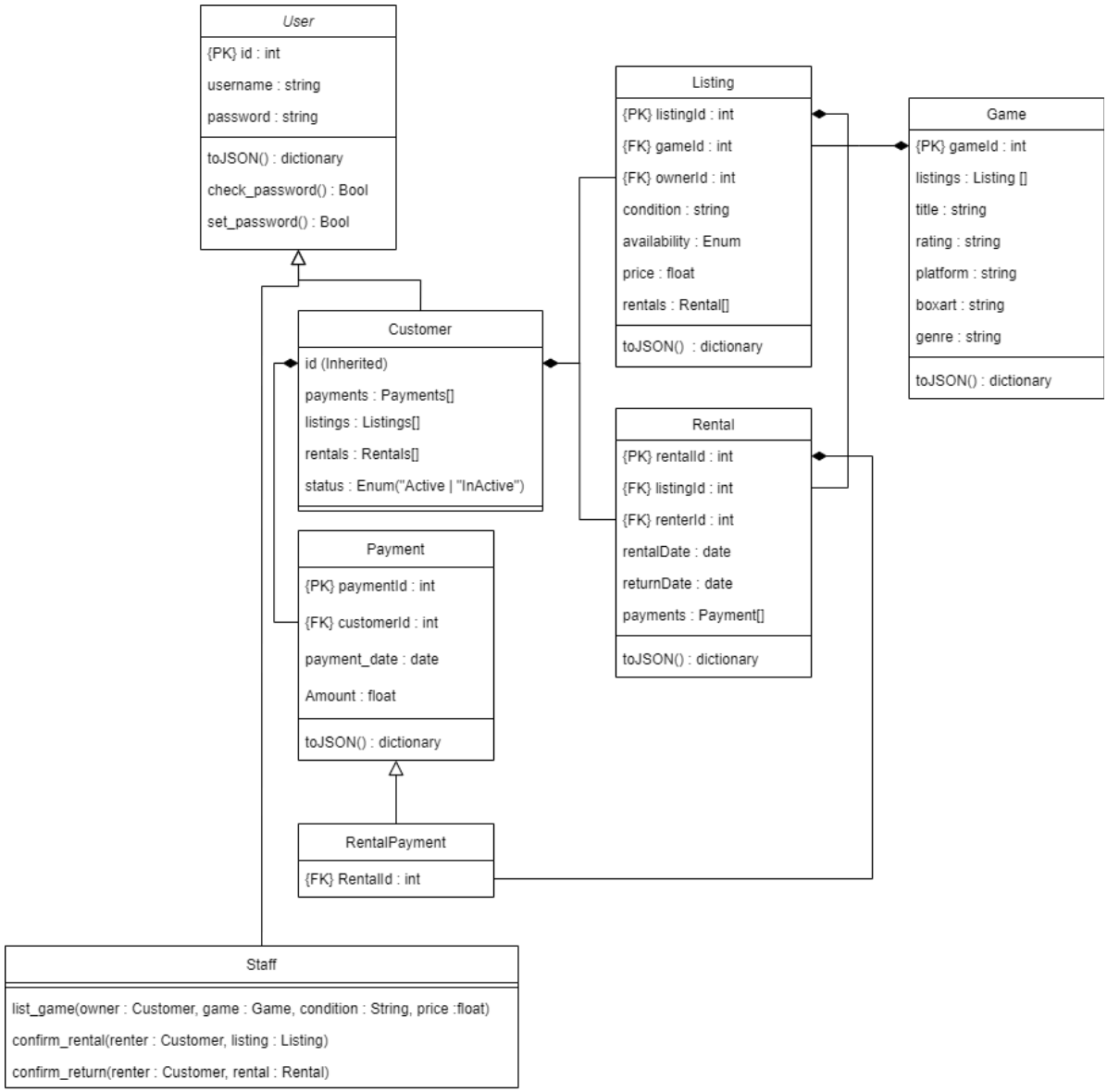
Pre-conditions: What state should the app already be in, what user class the user should be authenticated as.
Test Steps: Exactly what actions the user should perform on the UI
Test Criteria: What should the user see should they successfully complete the steps. How can the user visually verify on the UI that the desired operation was successful?

Question 1: Complete the test plan to specify how a feature that lets the user sell their game to the store might look.

Unit Test Testing

Unit testing allows us to verify the functionality of individual methods and functions in isolation**. All other dependencies are mocked**. Unit tests can be derived from our class diagram.

The following Class Diagram



Can have the following Unit Test Plan

Unit Test	Module Tested	Expected Output
test_new_user()	User(“bob”, “bobpass”)	newuser.username ==“bob”
test_toJSON()	User.to_json()	{ id:None, “username”:”bob” }
test_hashed_password()	User.set_password(“mypass”)	newuser.password != “mypass”

test_check_password()	User.check_password(“mypass”)	True

Notice how the modules tested are externally idempotent, that is they can be run multiple times without affecting any state external to the application code. eg database (Changes to memory is ok)

Question 2 : Can a unit test for Staff.create_listing() be made? Update the table accordingly.

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()		
test_staff_return_rental()		

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

Question 3: Complete the table for confirming rental and returns.

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>,
			“ownder”:<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 listings	GET	/listings	?platform=< NSW |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>
				}
				}
				]

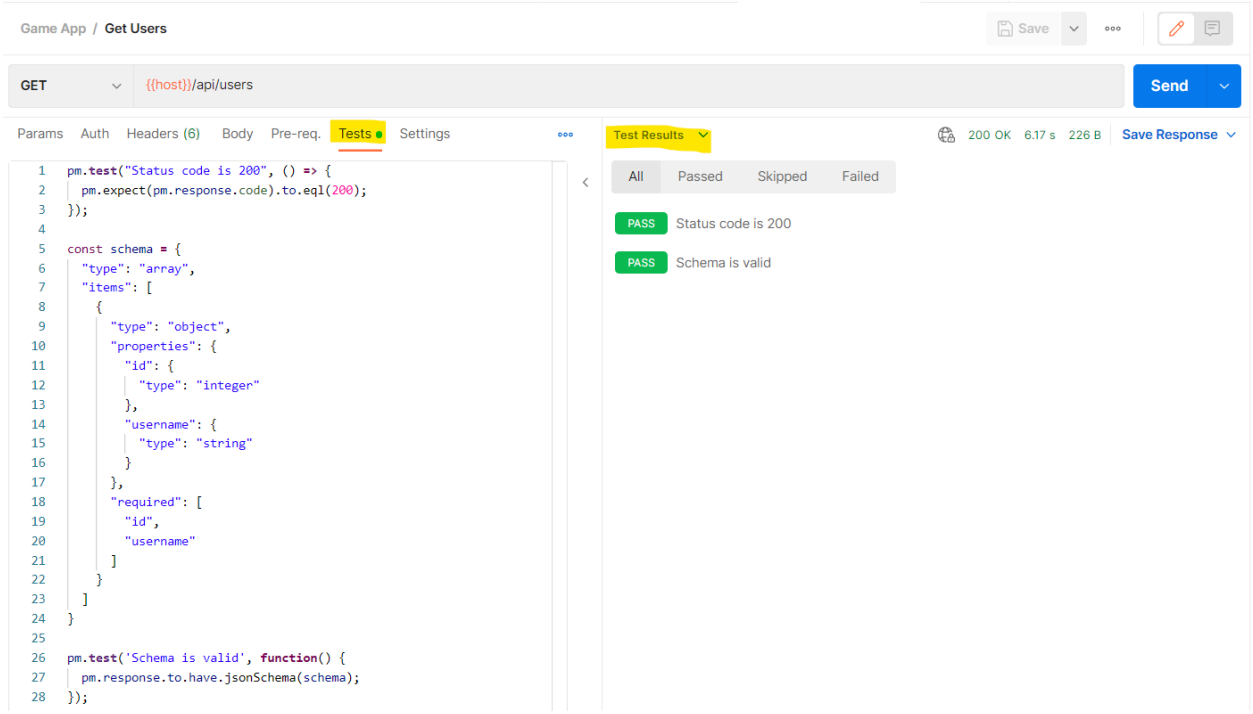
(Staff)
Make Payment
	POST	/payment	Authorization: JWT <token>
			
			{
			“amount” :<float>
			}
				{
				“message”: “payment created”,
				“paymentId”: <id>
				}, 201

(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				

API Tests are made in postman scripts.



View the full [Postman Collection]

Question 3: Update the API spec adding a route for confirming the selling of a game to the store as a customer.