Comprehensive Testing Plan

1. Test Strategy Overview

In this game-based learning application, backend testing plays a critical role in ensuring that the system functions correctly, securely, and efficiently. Since the backend (built using Flask) serves as the core interface between the frontend game (Unity in C#), the authentication layer (Firebase), the database (PostgreSQL), and the LLM question-generation service, it is essential to validate all functionalities through comprehensive tests. The focus of testing is on the API endpoints responsible for user authentication, campaign management, achievement tracking, character management, question handling, stats tracking, and user profile management. These endpoints handle critical operations such as fetching data, updating records, deleting entries, and ensuring secure access control. Testing is carried out using Pytest, with a mock-based approach to isolate the route logic from database dependencies and external services like Firebase authentication. This allows us to verify the correct response structures and status codes, proper interaction with the database layer, secure and accurate token validation and robust handling of edge cases.

Test Type	Scope	Tools/Frameworks Used
Unit Testing	Route handlers, business logic	pytest, unittest.mock, Flask test client
Mock Testing	DB queries, Firebase Auth, Models	unittest.mock.patch
API Response Test	HTTP Status codes and JSON responses	FlaskClient

2. Critical Features Covered

- **Authorization:** This includes the management of the signup and login routes, ensuring that users can securely create accounts and log in. Additionally, token verification is performed with Firebase to authenticate users for each request, ensuring that only authorized users can interact with the system. Testing detects Firebase authentication errors, handles missing fields with suitable error responses, confirms successful user creation with valid payloads, and validates the verify_firebase_token decorator. The decorator gets verified for situations involving valid tokens, invalid tokens, missing headers, and unregistered users.
- **Campaigns:** This includes fetching campaigns by the user to ensure that the right campaigns are linked to the right users. Updating campaign information ensures that changes to campaign attributes are correctly applied. Deleting campaigns is also tested to ensure that the system properly removes campaigns when required. The functionality of retrieving a single campaign is validated to ensure accurate campaign

data retrieval. Finally, restarting campaigns is tested to ensure that campaigns can be reset to their initial state.

- **Achievements:** This includes fetching all achievements associated with a user to validate that the correct achievements are linked. Adding new achievements is tested to ensure that new achievements can be correctly created and stored in the system. Additionally, the deletion of achievements is validated to ensure that users can properly remove achievements when needed.
- **Characters:** This includes retrieving a user's charcater information, changing character attributes, deleting characters, and generating new characters associated with a user with customization options (modelID, hardID, robeID, bootID). Tests verify that the database interacts correctly and that errors involving missing fields or nonexistent characters are handled.
- **Questions:** With error handling for missing or invalid data, this includes retrieving answers, deleting questions, fetching unanswered questions, creating questions individually and in batches (including file-based creation via the LLM service), answering questions (updating gotCorrect), incrementing wrong attempts, and retrieving questions by campaign or difficulty. The ability to load PDF content, create QA pairs with predetermined levels of difficulty, run an entire QA session across easy, medium, and hard levels, and skip malformed outputs are all tested for the LLM-based question generation service.
- **Stats:** This includes retrieving and updating player stats (such as attack, hp, mana, and affinity), creating stats, retrieving and assigning spells (four spells are required), deleting spells, and replenishing mana. Error handling for missing data, invalid payloads, and non-existent records is also included.
- **User Profile Mangement:** This includes deleting a user, retrieving a list of all users, updating user attributes (like screenName), retrieving a single user's details (like userID, screenName, and createdAt), and handling errors for users that don't exist.

3. Test Dataset Used

Test Context	**Sample Input / Mock Data**
Firebase User Token	{ "uid": "test-user" }
Campaign Mock Data	{ "campaignID": 1, "title": "Test Campaign", ... }
Achievement Mock Data	{ "achievementID": 101, "title": "First Win", "description": "Completed Level 1" }
Signup Payload	{ "email": "test@example.com", "password": "secret" }
Character Mock Data	{ "characterID": 1, "userID": "test_user_id", "modelID": 1, "hairID": 2, "robeID": 3, "bootID": 4 }
Question Mock Data	{ "questionID": 1, "campaignID": 1,

	"difficulty": "easy", "questionStr": "What is 2+2?", "gotCorrect": False, "wrongAttempts": 0 }
Answer Mock Data	{ "answerID": 1, "questionID": 1, "answerStr": "4", "isCorrect": True }
Batch Question Payload	[{ "campaignID": 1, "difficulty": "easy", "questionStr": "What is 2+2?", "answers": [{"answerStr": "4", "isCorrect": True}, {"answerStr": "5", "isCorrect": False}, ...] }]
PDF Mock Data	“test.pdf" file with content "Sample content"
QA Generation Mock Output	"Q1: What is X? (Medium)\nA) A\nB) B\nC) C\nD) D\nAnswer: C"
Player Stats Mock Data	{ "campaignID": 1, "attack": 10, "hp": 50, "mana": 30, "affinity": "fire" }
Player Spells Mock Data	{ "playerspellID": 1, "spellID": 1, "playerID": 1 }
Spell Mock Data	{ "spellID": 1, "spellName": "Fireball", "spellElement": "fire" }
Assign Spells Payload	{ "campaignID": 1, "spellIDs": [1, 2, 3, 4] }
Mana Replenish Payload	{ "manaAmount": 20 }
User Mock Data	{ "userID": "user1", "screenName": "TestUser", "createdAt": "2025-01-01" }

4. Expected vs Actual Results Table

Test Case	**Input/Request**	**Expected Output**	**Actual Output**	**Pass/Fail**
GET /campaigns/<uid>	Valid UID	200 OK with list of campaigns	200 status code with correct data	Pass
GET /campaigns/<uid>	Invalid UID (no campaigns)	200 OK, empty list	200 status code	Pass
PUT /campaigns/update/<id> (found)	Valid update body	200 OK, 'Campaign updated successfully'	200 status code	Pass
PUT /campaigns/update/<id> (not found)	ID not in DB	404 'Campaign not found'	404 status code	Pass
DELETE /campaigns/delete/<id>	Valid ID	200 'Campaign	200 status code	Pass

(found)		deleted successfully'		
DELETE /campaigns/delete/<id> (not found)	Invalid ID	404 'Campaign not found'	404 status code	Pass
GET /campaigns/single/<id> (found)	Valid ID	200 OK + campaign details	200 status code	Pass
GET /campaigns/single/<id> (not found)	Invalid ID	404 'Campaign not found'	404 status code	Pass
PATCH /campaigns/<id>/restart (found)	Valid ID	200 'Campaign restarted successfully'	200 status code	Pass
PATCH /campaigns/<id>/restart (not found)	Invalid ID	404 'Campaign not found'	404 status code	Pass
GET /achievements/<uid>	Valid UID	200 OK, list of achievements	200 status code	Pass
POST /achievements/<uid>	Valid payload	200 'Achievement added successfully'	200 status code	Pass
DELETE /achievements/<id> (found)	Valid ID	200 'Achievement deleted successfully'	200 status code	Pass
DELETE /achievements/<id> (not found)	Invalid ID	404 'Achievement not found'	404 status code	Pass
POST /signup	Valid payload	201 'User signed up successfully'	201 status code	Pass
POST /signup	Missing fields	400 'Invalid signup data'	400 status code	Pass
POST /characters/create	Valid payload: {"userID": "test_user_id", "modelID": 1, ...}	201 with character data	201	Pass
POST /characters/create	Missing fields: {"userID": "test_user_id"}	400 "Missing required fields"	400	Pass
GET /characters/	Valid UID	200 with	200	Pass

		character data		
GET /characters/	Invalid UID	404 "Character not found"	404	Pass
PUT /characters/update/	Valid ID, {"modelID": 5, "hairID": 6}	200 "Character updated successfully"	200	Pass
PUT /characters/update/	Invalid ID	404 "Character not found"	404	Pass
DELETE /characters/delete/	Valid ID	200 "Character deleted successfully"	200	Pass
DELETE /characters/delete/	Invalid ID	404 "Character not found"	404	Pass
Verify Firebase Token	Valid token, registered user	200 with {"uid": "test_user"}	Matches expected	Pass
Verify Firebase Token	No Authorization header	401 "Missing token"	401	Pass
Verify Firebase Token	Invalid token	403 "Invalid token"	403	Pass
Verify Firebase Token	Valid token, unregistered user	403 "User not registered"	403	Pass
GET /questions/	Valid campaignID	200 with question list	200	Pass
PUT /questions/answer/	Valid ID, {"gotCorrect": True}	200 "Answer recorded successfully"	200	Pass
PUT /questions/answer/	Invalid ID	404 "Question not found"	404	Pass

Batch Create Questions	Valid payload with 1 question	201 "Questions and answers created successfully"	201	Pass
Batch Create Questions	Missing fields in payload	400 "Missing required fields"	400	Pass
GET /questions/question/	Valid ID	200 with question data	200	Pass
DELETE /questions/delete/	Valid ID	200 "Question deleted successfully"	200	Pass
GET /questions/answers/	Valid questionID	200 with answer list	200	Pass
PUT /questions/wrong_attempt/	Valid ID	200 "Wrong attempt recorded"	200	Pass
GET /questions/unanswered	No input (authenticated)	200 with unanswered question list	200	Pass
GET /questions/difficulty/	Valid difficulty (e.g., "easy")	200 with filtered question list	200	Pass
POST /questions/create	Valid file (test.pdf), campaignID	201 "Questions created"	201	Pass
GET /stats/	Valid campaignID	200 with stats data	200	Pass
GET /stats/	Invalid campaignID	404 "Player stats not found"	404	Pass
PUT /stats/update/	Valid ID, {"attack": 15, "hp": 60}	200 "Player stats updated successfully"	200	Pass
PUT /stats/update/	Invalid ID	404 "Player stats not	404	Pass

		found"		
GET /stats/spells	No input (authenticated)	200 with spell list	200	Pass
POST /stats/assign_spells	Valid payload: {"campaignID": 1, "spellIDs": [1, 2, 3, 4]}	200 "Spells assigned successfully"	200	Pass
POST /stats/assign_spells	Invalid payload: {"campaignID": 1, "spellIDs": [1, 2]}	400 "Request must contain 'campaignID' and exactly 4 'spellIDs'"	400	Pass
GET /stats/player_spells/	Valid playerID	200 with player spell list	200	Pass
GET /stats/player_spells/	Invalid playerID	404 "No spells found for this player"	404	Pass
POST /stats/create	Valid payload: {"campaignID": 1, "attack": 10, ...}	200 "Player stats created successfully"	200	Pass
DELETE /stats/player_spells/delete/	Valid ID	200 "Spell removed successfully"	200	Pass
DELETE /stats/player_spells/delete/	Invalid ID	404 "Spell not found"	404	Pass
POST /stats/spells/create	Valid payload: {"spellName": "Fireball", ...}	200 "Spell created successfully"	200	Pass
PATCH /stats/replenish_mana/	Valid ID, {"manaAmount": 20}	200 "Mana replenished successfully"	200	Pass
PATCH /stats/replenish_mana/	Missing manaAmount	400 "Missing manaAmount"	400	Pass
PATCH /stats/replenish_mana/	Invalid ID	404 "Player stats not	404	Pass

		found"		
GET /users/	Valid userID	200 with user data	200	Pass
GET /users/	Invalid userID	404 "User not found"	404	Pass
DELETE /users/	Valid userID	200 "User deleted successfully"	200	Pass
DELETE /users/	Invalid userID	404 "User not found"	404	Pass
PUT /users/update/	Valid userID, {"screenName": "NewName"}	200 "User updated successfully"	200	Pass
PUT /users/update/	Invalid userID	404 "User not found"	404	Pass
GET /users/	No input (authenticated)	200 with user list	200	Pass
GET /users/	No users in DB	200 with empty list	200	Pass
Load PDF for QA	test.pdf with "Sample content"	Non-empty list with "Sample content"	Matches expected	Pass
Create QA	Context, difficulty "Medium"	Valid QA string with 4 options, correct answer	Matches expected	Pass
Run QA Session	PDF path, all difficulties, campaignID	List of 3 QA pairs (easy, medium, hard)	Matches expected	Pass
Run QA Session (Invalid Output)	PDF path, malformed LLM output	Empty list (skips invalid QA)	Matches expected	Pass

5. Test Validation Plan

- **API correctness:** All routes return correct status codes and expected JSON responses under all conditions (success, error, edge cases), including character, question, stats, and user data structures, as well as LLM-generated QA pairs.
- **Error handling:** All 404/400/401/403 cases are handled and tested with mocks for database not found, invalid data, missing fields, invalid tokens, unregistered users, malformed LLM outputs, and specific constraints.
- **Authentication handling:** All protected routes tested with mock Firebase tokens and mocked User objects, with the verify_firebase_token decorator validated for valid tokens, missing headers, invalid tokens, and unregistered users.
- **Mocked DB interaction:** All DB reads/writes are fully mocked using unittest.mock, ensuring tests are isolated from the actual DB for characters, questions, stats, and users.
- **Reusability and Scalability:** Structure allows adding more tests easily without touching core logic, applicable to all tested features.
- **Question Generation:** Validates PDF content extraction, LLM output parsing into structured QA pairs with exactly four answers and one correct option, handling of all difficulty levels, and robustness against invalid outputs by skipping them.
- **Stats-Specific Validation:** Ensures correct assignment of exactly four spells and accurate mana replenishment logic.

Testing Plan for Front-end testing

The objective of our testing is to verify that the core system functionalities of the GameManager class work as intended by developing and executing a comprehensive set of unit and integration tests in Unity.

Our validation plan was to use mocked dependencies for external systems like LevelManager, InterfaceManager, DialogueManager, etc., to isolate the GameManager logic. We also used black box testing to focus on expected inputs and outputs from public methods as well as white box testing to validate the internal game transitions for eg., isPaused, gametime etc. We used Unity's WaitForSeconds() for coroutine-based behavior validation.

Test Cases:

Test Case Name	Description	Test Data Set / Setup	Expected Result	Status
ResetGameState_Resets	Verify	gameTime = 10f,	gameTime = 0,	Pass

TimersAndStates	ResetGameState resets timers and game state values.	shownIntro = true, isPaused = false	isPaused = true, shownIntro = false	
FreezeAndUnFreezeGame_TogglesIsFrozen	Verify freeze and unfreeze toggle game state.	isPaused = false, call FreezeGame and then UnFreezeGame	After freeze: isPaused = true, IsFrozen = true After unfreeze: IsFrozen = false	Pass
PauseAndResumeMovement_ChangesPauseState	Verify PauseMovement and ResumeMovement methods change pause state.	isPaused = false, call PauseMovement then ResumeMovement	After pause: isPaused = true After resume: isPaused = false	Pass
CanRareSpawn_ReturnsTrueAfterDelay	Verify rare spawn only allowed after delay.	Initial gameTime = 0 rareSpawnDelay = x	First call: true, next call: false, after delay: true. CanRareSpawn returns true after delay	Pass
CanNPCSpawn_ReturnsTrueAfterDelay	Verify NPC spawn condition is based on delay.	npcSpawnDelay = 5f, gameTime = 0	First call: true, next call: false, after delay: true. CanNPCSpawn returns true after delay	Pass
CanEnemySpawn_ReturnsTrueAfterDelay	Verify enemy spawn condition is based on delay.	enemySpawnDelay = 5f, increment gameTime accordingly	First call: true, next call: false, after delay: true. CanEnemySpawn returns true after delay	Pass
ResumeAfterDelay_ResumesMovement	Ensure game resumes movement after short delay.	PauseMovement, enemyCount = 0, call ResumeAfterDelay(0.1)	Game resumes (isPaused = false) after delay	Pass