# **Game Library API, Spring Boot plus MongoDB**

## 1. Overview

A REST API to manage players, games, player game status, and named collections. Persistence is MongoDB. Validation uses Jakarta Bean Validation. Logging is via SLF4J. Unit tests use JUnit 5 and Mockito. The API returns ResponseEntity for explicit HTTP status and body control

## 2. Tech stack

* Java 17 plus Spring Boot 3.x for web and validation.
* Spring Data MongoDB, repositories extend MongoRepository for CRUD [2, 3].
* Jakarta Bean Validation annotations on records, validated in controllers with @Valid [4].
* JUnit 5 and Mockito for unit testing with MockitoExtension.

## 3. Configuration

* Mongo connection, application.yml  
  spring.data.mongodb.uri: mongodb://localhost:27017/game\_library
* Default port 8080 unless overridden.

## 4. Domain model

All domain types are Java records stored as MongoDB documents.

### Player [package com.anas.gameLibrary.player]

* + Fields, id, username, email, birthDate.
  + Constraints, @NotBlank id and username, @Email @NotBlank email, @Past birthDate. Violations raise MethodArgumentNotValidException that the handlers convert to 400 with details.

### Game [package com.anas.gameLibrary.game]

* + Fields, id, title, genre, platform, releaseDate.
  + Constraints, @NotBlank id, title, genre, @NotNull platform, @PastOrPresent releaseDate.
  + Platform enum, PC, PS5, XBOX, SWITCH, ANDROID, IOS.

### GameCollection

* + Fields, id, name, playerId, gameIds.
  + Constraints, @NotBlank id, name, playerId, @NotEmpty gameIds.

### PlayerGame

* + Fields, id, playerId, gameId, status.
  + Constraints, @NotBlank playerId and gameId, @NotNull status.
  + GameStatus enum, NOT\_STARTED, PLAYING, COMPLETED, ABANDONED.

## 5. Persistence layer

* PlayerRepository extends MongoRepository<Player, String> with findByUsername(String).
* GameRepository extends MongoRepository<Game, String>.
* GameCollectionRepository extends MongoRepository<GameCollection, String> with findByPlayerId(String).
* PlayerGameRepository extends MongoRepository<PlayerGame, String> with findByPlayerId, findByGameId, findByPlayerIdAndGameId.  
  Spring Data MongoDB generates implementations and basic queries from method names.

## 6. Services, business rules

### PlayerService

* + Standard CRUD. deletePlayer checks existence, returns boolean, logs outcomes.

### GameService

* + Standard CRUD, delete returns boolean, logs outcomes.

### GameCollectionService

* + CRUD plus mutation helpers, addGameToCollection(collectionId, gameId) adds if absent, removeGameFromCollection removes if present. Both return Optional updated document.

### PlayerGameService

* + Creates and updates links between players and games, ensures uniqueness, queries games by status for a player, queries players for a game, updates status, safe delete with existence check.
  + Also derives distinct ids and fans out to repositories as needed.

## 7. REST API

Paths, verbs, and typical responses. All endpoints return JSON and use ResponseEntity for clear statuses [1, 2, 13].

### 7.1 Players, /api/players

* GET /api/players returns 200 with list, or 204 when empty.
* GET /api/players/{id} returns 200 with entity, or 404 when missing.
* POST /api/players body is a Player, returns 201 or 400 if validation fails.
* PUT /api/players/{id} body is a Player, returns 200 with updated, or 404 if id not found.
* DELETE /api/players/{id} returns 204 on success, 404 if not found.

Sample requests are in players.http.

### 7.2 Games, /api/games

* GET /api/games returns 200 list or 204 if empty.
* GET /api/games/{id} returns 200 entity or 404.
* POST /api/games create with validation.
* PUT /api/games/{id} update with validation.
* DELETE /api/games/{id} returns 204 or 404.

Samples in games.http.

### 7.3 Collections, /api/collections

* GET /api/collections returns all collections.
* GET /api/collections/{id} returns one or 404.
* GET /api/collections/player/{playerId} returns collections for a player.
* POST /api/collections create.
* PUT /api/collections/{id} update.
* PUT /api/collections/{collectionId}/add/{gameId} add a game to collection, 200 or 404 if collection missing.
* PUT /api/collections/{collectionId}/remove/{gameId} remove a game, 200 or 404.
* DELETE /api/collections/{id} returns 204 or 404.

Samples in collections.http.

### 7.4 Player games, /api/player-games

* GET /api/player-games/player/{playerId} list player’s entries.
* GET /api/player-games/game/{gameId} list entries for a game.
* POST /api/player-games create an entry.
* PUT /api/player-games/status?playerId=..&gameId=..&status=.. update status, returns 200 with updated or 404 when relation missing.
* GET /api/player-games/status/{playerId}?status=COMPLETED returns game ids for a player filtered by status.
* DELETE /api/player-games/{id} returns 204 or 404.

Samples in playersGames.http.

## 8. Validation and error handling

* All controllers use @Valid on inputs so Jakarta Bean Validation runs. Typical constraints, @NotBlank, @Email, @Past, @PastOrPresent, @NotNull, @NotEmpty [4].
* Two global handlers exist under com.anas.gameLibrary.exception:
  + ValidationExceptionHandler, handles MethodArgumentNotValidException, returns 400 with a map of field -> message.
  + GlobalExceptionHandler, also handles MethodArgumentNotValidException but formats as a single joined string, plus handles IllegalArgumentException to 400. You should keep only one style for consistency. Both are annotated with @ControllerAdvice which is the correct mechanism for centralized exception handling in Spring MVC [1].  
    If both remain, Spring will pick a matching handler method, but message shape could vary across endpoints. Pick one format and remove the other.

## 9. Logging

* Service and controller layers log at info for normal operations and warn for not found or validation issues. SLF4J, LoggerFactory.getLogger(...).

## 10. Test suite

* GameServiceTest and PlayerServiceTest use JUnit 5 and Mockito with @ExtendWith(MockitoExtension.class), @Mock, @InjectMocks, when(...), verify(...), assert.... This is the standard approach with JUnit 5 extensions and Mockito [5]. Assertions rely on JUnit 5 user guide semantics.
* GameLibraryApplicationTests includes a context load smoke test with @SpringBootTest.

## 11. Sample data and quick start

* Seed JSONs, players.json, games.json, collections.json, playersGames.json. Use them via Postman or the included .http files in IntelliJ or VS Code REST Client. Or import into Mongo using mongoimport if you want initial collections:
  + mongoimport --uri mongodb://localhost:27017/game\_library --collection players --file players.json --jsonArray
* Start MongoDB locally on 27017, then run the app:
  + ./mvnw spring-boot:run or run from your IDE.
* Health check, GET http://localhost:8080/api/games should return list from your DB if you imported, or 204 if empty.

## 12. API rules and status codes

* 200 OK for successful GET or update, 201 Created for successful POST with body, 204 No Content for successful delete or empty list retrieval where you deliberately chose 204, 400 Bad Request on validation or illegal arguments, 404 Not Found when an id does not exist. This is consistent with Spring MVC patterns and ResponseEntity usage.

## 13. Project layout, high level

* .../game package, Game, Platform, repository, service, controller.
* .../player package, Player, GameCollection, PlayerGame, enums, repositories, services, controllers.
* .../exception package, exception handlers.
* src/test/java/... has JUnit tests for services.

## 14. Security and hardening, next steps

* Add input id normalization, for example reject mismatched id in body versus path on PUT, or ignore body id and enforce path id.
* Consistent error format, choose the map shape or the joined string, document it in this section, for example { "errors": { "email": "must be a well formed email address" } }.
* Add OpenAPI spec with springdoc, expose Swagger UI.
* Add unique constraints where appropriate, for example Player.username should be unique. With Mongo, enforce via an index.
* Consider referential checks in services, for example, ensure a PlayerGame cannot be created if either playerId or gameId does not exist.
* Add pagination for list endpoints.
* Add authentication and authorization later, for example Spring Security, then restrict collection changes to owning player.