**🧠 Project Name: Game Library Management System (Spring Boot + MongoDB)**

A RESTful backend system built with **Java Spring Boot** that manages games, players, collections, and game statuses per player. It uses **MongoDB** for persistence and supports full CRUD via a structured REST API.

**🔧 1. Technologies and Programming Languages**

* **Languages**:  
  Java (backend), JSON (test data), HTTP (API testing)
* **Frameworks & Libraries**:  
  Spring Boot (main framework), Spring Data MongoDB (data access), JUnit (testing)
* **Database**:  
  **MongoDB** (NoSQL, document-based), configured via application.yml
* **Testing Tools**:
  + JUnit for unit/integration tests
  + REST Client (\*.http files used in VS Code or IntelliJ) for manual API testing
* **Build Tool**:  
  Maven

**🧱 2. Architecture and Structure**

Follows a **layered architecture** using **Spring MVC** with MongoDB integration.

* **Model Layer**:  
  Domain classes Game, Player, GameCollection, PlayerGame
* **Repository Layer**:  
  Interfaces like GameRepository, PlayerRepository, etc., extend MongoRepository, abstracting MongoDB document operations
* **Service Layer**:  
  Contains business logic such as:
  + Validation (e.g. unique usernames)
  + Game-player relationship management
  + Collection modification
* **Controller Layer**:  
  REST controllers exposing CRUD endpoints for each resource using @RestController
* **Exception Handling**:
  + ValidationExceptionHandler: Handles input violations (e.g., blank email)
  + GlobalExceptionHandler: Catches and formats custom or unhandled exceptions
* **Testing Layer**:  
  Unit and integration tests validating service and application logic

**📦 3. Backend Features**

**A. 🎮 Games Module**

* Full CRUD support
* Fields:
  + id (string)
  + title
  + genre
  + platform (enum: PC, PS5, XBOX)
  + releaseDate
* Enum used for platform: Platform.java
* Can be referenced by collections or players

**B. 👤 Players Module**

* Register and manage players
* Fields:
  + id
  + username (unique)
  + email (unique)
  + birthDate
* Validation enforced at the service layer
* Duplicates rejected with meaningful error messages

**C. 📚 Collections Module**

* Represents curated lists like "Favorites", "Backlog"
* Each collection has:
  + id
  + name
  + playerId (reference)
  + gameIds (list of referenced games)
* Managed via GameCollectionController

**D. 🔄 Player-Game Relationship Module**

* Core entity: PlayerGame (composite ID = playerId + gameId)
* Tracks each player’s game status:
  + NOT\_STARTED
  + PLAYING
  + COMPLETED
* APIs to:
  + Set or update status
  + Fetch all statuses for a player

**E. 🛡️ Validation and Exception Handling**

* **ValidationExceptionHandler.java**:
  + Centralizes form and field validation error formatting
* **GlobalExceptionHandler.java**:
  + Catches logical/operational errors (e.g. not found, bad request)

**💾 4. Database Structure (MongoDB)**

Despite early use of JSON for test seeding, the current implementation is backed by MongoDB:

**Collections & Structure**

* games:
  + Stores all games with metadata
* players:
  + Player accounts and info
* gameCollections:
  + Lists of game IDs grouped under a player
* playerGames:
  + Status tracking of specific games per player

**JSON Test Files (previously used or still useful for mocking)**

* players.json – sample users
* games.json – test game catalog
* collections.json – predefined game groups
* playersGames.json – initial status assignments

**🌐 5. API Access (via REST Controllers)**

Each module has its own controller with clean, RESTful endpoints:

* /games
  + GET all, GET by ID, POST, PUT, DELETE
* /players
  + Same structure for user accounts
* /collections
  + Manage player game collections
* /playersGames
  + Update and retrieve game statuses per player

APIs return JSON, use standard HTTP status codes (200, 201, 404, 400), and include proper error messages.

**🧪 6. Testing Approach**

* **Unit Tests**:
  + GameServiceTest, PlayerServiceTest for business logic
* **Integration Test**:
  + GameLibraryApplicationTests to ensure application context loads properly
* **Manual HTTP Testing**:
  + .http files for every module simulate API requests via REST Client

**✅ Summary**

This backend is a complete, scalable, and modular REST API system designed using:

* Spring Boot + MongoDB
* RESTful controller structure
* Robust error handling and validation
* Data referencing (not joins) across entities
* Fully testable with both unit and integration strategies