Online Video Games Store Domain:

The system is showcasing an online video game store like Steam or GOG.

Game Model: A model for creating a video game with the following attributes and its validators:

1. String id: Each game has a unique ID.
   Validators:
   @NotEmpty(message = "Game ID cannot be empty.")
   Ensures the id won't be empty because each game must be uniquely identified in the system. Some games might have identical names or different versions.
   @Pattern(regexp = "^GA-(\\d){10}$", message = "Game ID must start with 'GA-' followed by 10 digits.")
   Ensures The id has a unique and identifiable format, made it long because there's millions of games.

2. String title: Each game has a title/name.
   Validators:
   @NotEmpty(message = "Title cannot be empty.")
   Ensures The game has a title to be identifiable and searchable by the users.
   @Length(min = 5, message = "Title must be at least 5 characters long.")
   Ensures the game title would have at least 5 letters.

3. String developerStudio: Each game has it's developing studio.
   Validators:
   @NotEmpty(message = "Game studio cannot be empty.")
   Ensures The game has its game studio in it's info, this can allow users to search games by their favorite game studio or to avoid certain scammers uploading a cheap game using a huge title.
   @Length(min = 5, message = "Title must be at least 5 characters long.")

4. String developerStudioUrl: Each game has it's developing studio.
   Validators:
   @NotEmpty(message = "Game studio URL cannot be empty.")
   Ensures The game studio has a URL to its official page so fans can check news there or look or merchandise for a specific game from the studio.
   @Length(min = 25, message = "URL must be at least 25 characters long.")
   A step to check the validity of the URL
   @URL(message = "Developer Studio URL must be a valid URL.")
   Check if the URL format is correct by checking if it begins with https:/http:.

5. String mainGenre: Each game has a main genre that users might search for or avoid.

    Validators:

    @NotEmpty(message = "Main Genre cannot be empty.")
    Ensures The game has a defined genre. It's a must since majority of players will be looking for games based its genres, each to their own taste, some might want to block horror games or some might only care for RPGs.
    @ Pattern (regexp = "^(Action|Horror|Adventure|Indie|RPG)$", message = "Main Genre must be one of the following: Action, Horror, Adventure, Indie, RPG.")
    Only most known/popular genres are allowed into the system (so far), beside making it easy for users to search for a game aligning with their interests it'll allow for easier monitoring and moderating to divide games so each genre has it's own team of moderators.

6. ArrayList<Review> reviews: Each game has a list of user reviews, so users can express their opinions and filter games with bad reviews.

    Validators:

    @Null(message = "A game cannot be created with reviews.")
    Ensures The game will be entered into the system with only reviews added from the users later after they played it.
    @Valid
    Only validated review can be entered into the system, this ensures any review entered must pass any validation inside Review model.

7. Boolean hasMultiPlayers: Games can either have multiplayer capabilities or not, some users likes the former, others prefer the later.

    Validators:

    @NotNull (message = "Has multiplayer support cannot be null.")
    Ensures The game will indicate either it has multiplayer or not, since most players prefer it does they might want to filter if it doesn't, other might prefer to play alone.

8. double price: Each game has its price.

    Validators:

    @NotNull(message = "Game must have a price.")
    Ensures The game has a price. Ensures profit to studios, and another thing users might want to filter depending on their budget.
    @Positive(message = "Price must be a positive value.")
    In case a game's price it's considered free, it's fine. But a negative price doesn't make since, and probably illegal.

Review Model: A model for creating a review by the users to each game with the following attributes and its validators:

1. String id: Each game has a unique ID.

   Validators:

   @NotEmpty(message = "Review ID cannot be empty.")
   Ensures the id won't be empty. Making reviews will make it easier to implement a report system using ID to track irrelevant of offensive review by allowing the users to report them. The id will be sent to the report system (a future plan)
   @Pattern(regexp = "^RV-(\\d){10}$", message = "Review ID must start with RV -' followed by 10 digits.")
   Ensures the review id has a unique and identifiable format.

2. User user: Each review is written by user.

   Validators:

   @NotNull(message = "Review must have a corresponding user.")
   So the user name can be printed in the review, and to ban the user in case of misuse.

3. boolean hasPlayed: Only users who actually played the game can post a review, buying it is not even enough.

   Validators:

   @NotNull(message = "Has played must not be null.")
   To check whether the user played the game or not.
   @AssertTrue(message = "Only users who played the game can post a review)
   Prevents users from entering a review if they didn't have the time to play the game.

4. int score: Each review is has a score for it.

   Validators:

   @NotNull(message = "Score cannot be null.")
   So the user can score the game based on their enjoyment or other factors
   @PositiveOrZero(message = "only positive scores or a zero are allowed")
   Cannot give a game a negative response.
   @Range(min = 0, max = 10, message = "Score must be between 0 and 10.")
   0 to 10 is a valid scoring system, allowing for mixed opinions not just good or bad.

5. String comment: Each review is has a comment from user.

    Validators:

    @NotEmpty(message = "Comment cannot be empty.")
    So the user can express why they gave the corresponding score. And tell other players why they should or shouldn't get the game.
    @Length(min=10, max=1200, message = "comments must have at least 10 characters")
    This would both allow for brief comments like "good game" or a professional paragraph about the game.

User Model: A model for creating a user in the online video game store system, with the following attributes and its validators:

1. String id: Each user has a unique ID.

    Validators:

    @NotEmpty(message = "User ID cannot be empty.")
    Ensures the id won't be empty. This will give users unique identifiers in the system. Dealing with user ID is way more efficient than usernames. It'll allow users to change their usernames without losing their identity or information in the system since it's all linked to the ID instead. Allow users to go incognito and still be able to track their progress in the games.
    @Pattern(regexp = "^USR-(\\d){10}$", message = "Review ID must start with USR -' followed by 10 digits.")
    Ensures the user id has a unique and identifiable format.

2. String username: Each user has an identifier for comments and in games.

    Validators:

    @NotEmpty(message = "User ID cannot be empty.")
    Ensures the id won't be empty. This will give users unique identifiers in the system. Dealing with user ID is way more efficient than user names. It'll allow users to change their user names without losing their identity in the system. Allow users to go incognito and still be able to track their progress in the games.
    @Length(min=4,max16, message = "username must have between 4 and 16 characters")
    Ensures the user doesn't enter too short or too long username.

3. String password: Each user has a password to protect their accounts.

   Validators:

   @NotEmpty(message = "Enter a password.")
   Ensures the user account has a password to login into the system. Each account MUST have a password since personal information and billing information are within each account information.
   @Length(min=16 message = “password must have at least 16 characters”)
   Ensures the user doesn’t enter a password that’s too short, 16 is hard to crack in case of a brute force attack, or hard to guess in general.
   @Pattern (regexp = "^(?=.*[A-Za-z])(?=.*\\d)(?=.*[@$!%*#?&])[A-Za-z\\d@$!%*#?&]{16,}$", message = "password must include letters, numbers and a special character. Password must be al least 16 character.")
   Ensures the password has a variety of characters, making it harder to crack. This way we’re securing users accounts.

4. String email: Each user has an email to login, get news, alerts, and bills.

   Validators:

   @NotEmpty(message = "Enter an email.")
   Ensures the user account has an email, emails are the main way to login into the system and to communicate with the users about anything regarding actions relating to their account.
   @Length(min=15 message = “email must have at least 16 characters”)
   A step in ensuring the user is entering a valid email. @gmail.com as an example is 10 characters, the rest is the user’s email name.
   @Email(message = “is invalid”)
   Ensures the email has the @ before accepting it into the system.