VHS Rental Management System: A Retro Tech Project

Welcome to "Blast from the Past," our cutting-edge management system for a classic VHS rental experience. This project involves building a robust backend service to manage our inventory, users, and rentals.

📋 Technical Specifications

This document outlines the core requirements for the application's backend.

API Endpoints

The application must expose a RESTful API. The primary resources are **`VHS`**, **`User`**, and **`Rental`**.

URI Path	Description
`/api/vhs`	Exposes the VHS resource endpoints.
`/api/rentals`	Exposes the Rental resource endpoints.
`/api/users`	Exposes the User resource endpoints.

Data Model (JPA)

Model the core business domains as JPA entities:

- **VHS:** Represents a single VHS tape. Should include attributes like `title`, `genre`, and `releaseYear`.
- **User:** Represents a customer. Should include basic user information like `name` and `email`.
- **Rental:** Represents a transaction linking a `User` and a VHS. Should include `rentalDate`, `dueDate`, and `returnDate`.

Business Logic & Validation

- **Rental Constraints:** The system must prevent the same VHS tape from being rented by multiple users on the same date.
- **Due Dates & Fees:** Implement logic to calculate rental due dates and handle potential late fees upon return.

- **Input Validation:** Use Bean Validation on request objects (e.g., `RentalForm` to validate incoming data to the `RentalController`).
- **Exception Handling:** Implement exception handlers using `@ExceptionHandler` to catch and format all application exceptions into clear error responses.
- **Custom Error Messages:** Error messages returned by the REST controller should be customized using a `MessageSource`.

Core Technologies

- **Framework: Spring Boot 3.x**
- **Data Persistence: Spring Data JPA** (with Hibernate)
- **Database:** Prepopulate a database of your choice. **H2** (for development) or **PostgreSQL** are recommended.
- **Logging:** Use **SLF4J** with a **Logback** implementation for application logging.

✅ Deliverables

Source Code

The complete source code for the Spring Boot application, managed in a Git repository. The project must include at least one meaningful automated test for a key piece of functionality (e.g., a service method or controller endpoint).

Postman Collection

A Postman collection must be included in the root of the Git repository (`vhs-collection.json`). It should contain pre-configured requests to demonstrate the API's functionality, including:

- **VHS**
  - List
  - VHS Rent and Return option
- **Rentals**
  - List

Setup instructions

Provide setup instructions so that your technical interviewer can run your backend service, and database, and test it out. Do this in the README.md file.

🎬 Director's Cut (Bonus Challenges)

For those looking to go the extra mile, completing any of the following "special features" will earn you bonus points.

- **Package It for Prime Time (Dockerize the App)** Just like a VHS tape needs a sturdy case, modern applications need proper packaging. Create a `Dockerfile` to containerize the Spring Boot application, making it portable and easy to run anywhere.

- **Build the Storefront (Create a Frontend)** Every video store needs a welcoming front counter. Develop a simple frontend application (using a framework like React, Angular, or Vue.js) that consumes your API, allowing users to visually browse the VHS collection and manage rentals.

- **Add a "Special Feature" (Implement New Functionality)** Surprise us with a new feature, just like a hidden post-credits scene. Here are some ideas:

    - **User Reviews:** Allow users to rate and review VHS tapes.
    - **Waitlist:** Implement a system for users to get on a waitlist for popular, currently rented-out tapes.
    - **Advanced Search:** Add functionality to search or filter the collection by genre, actor, or release year.
    - **Recommendations**: Recommend VHS tapes based on users previous rentals

🛠️ Development Environment Setup

Recommended Software

- **Java JDK: JDK 17 or higher** (LTS versions like 17 or 21 are recommended).
- **Maven: 3.8.x or higher**.
- **Git:** Latest stable version.
- **IDE: IntelliJ IDEA** or any other modern IDE with good Spring support.
- **API Client: Postman** or a similar tool for API testing.

Installation Links

- **Java:** Adoptium Temurin
- **Maven:** Apache Maven Project
- **Git:** Official Git Website
- **IntelliJ IDEA Community:** JetBrains
- **Postman:** Official Postman Website

🚀 Project Initialization & Workflow

1. Generate Project via Spring Initializr

Use the **Spring Initializr** to bootstrap your application with the following dependencies:

- Spring Web
- Spring Data JPA
- Bean Validation
- H2 Database (or PostgreSQL Driver)
- Lombok (optional, for convenience)

https://start.spring.io/

2. GitHub Workflow

1. **Set up GitHub:** Create a personal GitHub account if you don't have one.
2. **Repo:** Create a new public repository on your account.
3. **Clone:** Clone your repository to your local machine.
4. **Submission:** Send us the URL to your repository when you are happy with the state of your assignment.

📚 Recommended Resources

- **Baeldung - Intro to Spring:** https://www.baeldung.com/spring-intro
- **Spring Boot Documentation:** https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
- **Manning - Spring Boot in Action:** https://www.manning.com/books/spring-boot-in-action