Proposal: Media Collection Management System

Emily Weed

Feb 2, 2025

Content

Problem Statement

As personal and institutional media collections grow, users require an efficient, searchable, and maintainable system to organize, track, and manage their assets. Existing solutions can be overly complex or unsuitable for diverse media types.

Solution Overview: The Media Collection Management System (MCMS) provides a modular foundation for cataloging books, movies, music, and more. It enables users to add, modify, delete, and search media items within a user-friendly system. It is designed with scalability in mind, allowing for future expansions such as loan management, user accounts, and external API integration.

System Requirements:

- CRUD (Create, Read, Update, Delete) operations on media items
- Search and filtering capabilities
- Modular and scalable codebase
- Console-based UI (current version) with the ability to evolve into a web-based or mobile interface

Functional Requirement Specification

- Users can add new media entries with title, creator (author/director/artist), year, genre, and format.
- Users can list all current media entries.
- Users can search media by title, genre, or format.
- Users can update existing entries.
- Users can delete media entries.
- System provides validation on user input.
- System is structured to allow easy future additions (e.g., user accounts, loans).

System Sequence Diagram

Scalable to additional operations such as "Loan Media," "Return Media."

User --> Menu --> Media Manager
| |-> Add Media
| |-> Update Media
| |-> Delete Media
| |-> Search/List Media

Activity Diagram

Start --> Display Menu --> [User Choice]

If Add Media --> Prompt User --> Validate Input --> Save to Collection --> Display Confirmation

If Update Media --> Search for Media --> Edit Fields --> Save Changes

If Delete Media --> Search for Media --> Confirm Deletion --> Remove

If Search/List Media --> Display Matching Results

Exit --> End

User Interface Specification

Current UI: Console-based interface using menus and prompts.

Possible Extensions:

- Web-based UI using React.js (future)
- Mobile app interface (future)
- API endpoints for third-party integrations (future)

Implementation Planning

Week 1-2

- Design system architecture and define database schema (media types, user data).
- Set up the development environment and start building the front-end

Week 3-4

- Implement media cataloging functionality (adding, editing, deleting items).
- Develop user account management (authentication, roles, permissions).
- Begin building the media search functionality.

Week 5-6

- Test search, loan tracking, and metadata population features.

Week 7-8

- Build wishlist functionality and user media collection management.
- Develop recommendations for media based on user preferences.
- Test all user-facing features and improve UI/UX.

Week 9-10

- Begin testing the system with real users, including librarians, rental store employees, and personal media collectors.
- Collect feedback and refine features based on input.
- Address any bugs or performance issues.

Week 11-12

- Conduct final testing, especially on media metadata integration.
- Finalize documentation and prepare user manuals.
- Prepare a demo for the final presentation.

System Architecture and System Design

Architecture:

- Modular Code Structure (Separate Media Item, Manager, and UI handling)
- MVC-Inspired Separation of Concerns (Planned for Web Version)

Design:

- Media Item Class: Represents individual media entries.
- Media Manager Class: Handles collection management and operations.
- Main Application File: Handles user interface and logic flow.

Algorithms and Data Model Implementation

Data Model:

- MediaItem Class with attributes (title, creator, year, genre, format).

Key Algorithms:

- Search by title/genre/format using case-insensitive matching.
- Update by locating a media item and overwriting attributes.
- Validation algorithms for ensuring correct user input types.

User Interface Design and Implementation

- Text-based menus.
- Clear prompts and minimal error messages.
- Designed for clarity and user-friendliness.

Future:

- Modern UI frameworks for enhanced experience (planned extensions).

Design of Tests

- Unit tests for add, update, delete, and search operations.
- Mock user input where applicable.
- Tests for invalid input scenarios.

Error Handling and Debugging

- Input validation with re-prompts for incorrect entries.
- Try-except blocks around critical sections to prevent crashes.
- Debugging conducted via step-through tracing and print logs.

Collaboration and Code Integration

- Version control via Git and GitHub
- Code was modularly developed for ease of integration.
- Version control was maintained with modular commits.
- Integration testing performed after each major feature addition.

Performance Optimization Plan

Current System:

- Optimized for small-medium collections (under 10,000 entries).

Future Optimization Plans:

- Migrate to database storage for very large collections.
- Indexing fields for faster search operations.
- Parallel processing for batch operations.

Maintenance and Support Strategy

- Clean, readable code with inline comments.
- Future maintenance includes adding persistence (file/database), enhancing UI, and expanding search capabilities.
- Documentation prepared for each class/module.