Bookify - Complete Jira Task Breakdown

🎯 Simple, Step-by-Step Development Plan

Current Status: ✅ Category CRUD Complete
Next Up: 👤 User Management

📊 How to Use This Guide

Task Structure:

- **Epic** = Big Feature (e.g., "User Management")
- **Story** = User-facing functionality (e.g., "User can register")
- **Task** = Technical work (e.g., "Create User model")
- **Subtask** = Small steps within a task

Priority Levels:

- 🔴 **Critical** - Must have, blocking other features
- 🟠 **High** - Important, needed soon
- 🟡 **Medium** - Nice to have, not urgent
- 🟢 **Low** - Future enhancement

Estimated Time:

- **XS** = 1-2 hours
- **S** = 2-4 hours
- **M** = 4-8 hours (half day to full day)
- **L** = 1-2 days
- **XL** = 2-5 days

✅ PHASE 0: COMPLETED

Epic: Category Management

☑ **BOOK-1:** Create Category Model

☑ **BOOK-2:** Create Category Controller

☑ **BOOK-3:** Create Category Service

☑ **BOOK-4:** Add Category Routes

☑ **BOOK-5:** Test Category CRUD Operations

Status: ✅ DONE! Great job! 🎉

🚀 PHASE 1: USER MANAGEMENT (START HERE!)

Epic: User Authentication & Authorization

Story 1: User Registration

As a new user, I want to register an account so I can access the platform

BOOK-10: Setup User Model 🔴 Critical | Size: M (4-6h)

Description:
Create MongoDB schema for users with all necessary fields

Acceptance Criteria:
- [ ] User schema created with fields: email, password, firstName, lastName, role
- [ ] Email field is unique and required
- [ ] Password is hashed using bcrypt (cost factor 12)
- [ ] Default role is 'user'
- [ ] Timestamps (createdAt, updatedAt) added
- [ ] isActive field (default: true)
- [ ] isEmailVerified field (default: false)

Technical Notes:
- Use mongoose schema
- Add pre-save hook for password hashing
- Add method to compare passwords
- Don't store plain text passwords!

Files to Create:
- models/User.js

BOOK-11: Create User Registration Endpoint 🔴 Critical | Size: M (4-6h)

Description:
Build API endpoint for user registration

Acceptance Criteria:
- [ ] POST /api/auth/register endpoint created
- [ ] Validates email format
- [ ] Validates password strength (min 8 chars, mixed case, numbers, special chars)
- [ ] Checks if email already exists
- [ ] Creates new user in database
- [ ] Returns success message (don't return password!)
- [ ] Returns 400 for validation errors
- [ ] Returns 409 if email already exists

Technical Notes:
- Use express-validator for validation
- Service layer handles business logic
- Controller handles HTTP requests/responses

Files to Create:
- controllers/authController.js
- services/authService.js
- routes/authRoutes.js
- middleware/validation/authValidation.js

Test Cases:
- Valid registration
- Duplicate email
- Invalid email format
- Weak password

BOOK-12: Add Email Verification (Optional for now) 🟡 Medium | Size: L (1-2 days)

Description:
Send verification email with OTP after registration

Acceptance Criteria:
- [ ] Generate 6-digit OTP
- [ ] Store OTP in database with expiry (10 minutes)
- [ ] Send email using nodemailer
- [ ] Create verify-email endpoint
- [ ] OTP expires after 10 minutes
- [ ] User can request new OTP

Technical Notes:
- Use nodemailer for email service
- Store email config in .env
- Consider using email templates

Files to Create:
- services/emailService.js
- templates/verificationEmail.html

Dependencies:
- npm install nodemailer

Story 2: User Login

As a registered user, I want to log in so I can access my account

BOOK-15: Setup JWT Configuration 🔴 Critical | Size: S (2-3h)

Description:
Configure JWT tokens for authentication

Acceptance Criteria:
- [ ] JWT_SECRET added to .env
- [ ] JWT_REFRESH_SECRET added to .env
- [ ] Access token expiry: 15 minutes
- [ ] Refresh token expiry: 7 days
- [ ] Helper functions to generate tokens
- [ ] Helper functions to verify tokens

Technical Notes:
- Use jsonwebtoken library
- Store secrets in .env (NEVER commit these!)
- Use strong random secrets (64+ characters)

Files to Create:
- utils/tokenUtils.js
- config/jwt.config.js

Dependencies:
- npm install jsonwebtoken

BOOK-16: Create Login Endpoint 🔴 Critical | Size: M (4-6h)

Description:
Build API endpoint for user login

Acceptance Criteria:
- [ ] POST /api/auth/login endpoint created
- [ ] Validates email and password
- [ ] Checks if user exists
- [ ] Verifies password using bcrypt
- [ ] Generates access token (JWT)
- [ ] Generates refresh token
- [ ] Returns tokens and user data (no password!)
- [ ] Sets refresh token as httpOnly cookie
- [ ] Returns 401 for invalid credentials
- [ ] Returns 403 if account is inactive

Technical Notes:
- Don't reveal whether email or password is wrong (security)
- Use authService.login() method
- Store refresh token in database

Response Format:
```
{
  "success": true,
  "data": {
    "accessToken": "eyJhbGc...",
    "user": {
      "id": "...",
      "email": "...",
      "role": "user"
    }
  }
}
```

Files to Update:
- controllers/authController.js
- services/authService.js
- routes/authRoutes.js

Test Cases:
- Valid login
- Invalid email

- Invalid password
- Inactive account

BOOK-17: Create Authentication Middleware 🔴 Critical | Size: M (4-6h)

Description:
Middleware to protect routes requiring authentication

Acceptance Criteria:
- [ ] Extract token from Authorization header
- [ ] Verify token using JWT_SECRET
- [ ] Decode user info from token
- [ ] Attach user to req.user
- [ ] Return 401 if token missing
- [ ] Return 401 if token invalid/expired
- [ ] Allow request to continue if valid

Technical Notes:
- Token format: "Bearer <token>"
- Handle token expiry gracefully
- Add helpful error messages

Files to Create:
- middleware/auth.middleware.js

Usage Example:
router.get('/profile', authenticate, getProfile);

Story 3: Token Refresh

As a user, I want my session to persist without logging in repeatedly

BOOK-20: Create Refresh Token Endpoint 🔴 Critical | Size: M (5-7h)

Description:
Allow users to refresh their access token

Acceptance Criteria:
- [ ] POST /api/auth/refresh endpoint created
- [ ] Reads refresh token from httpOnly cookie
- [ ] Verifies refresh token
- [ ] Checks if token exists in database
- [ ] Generates new access token
- [ ] Rotates refresh token (issue new one)
- [ ] Invalidates old refresh token
- [ ] Returns new access token
- [ ] Sets new refresh token cookie

Technical Notes:
- Refresh token rotation prevents theft
- Store refresh tokens in User model or separate collection
- Implement token family detection (optional, advanced)

Files to Update:
- controllers/authController.js
- services/authService.js
- models/User.js (add refreshTokens array)

Story 4: User Logout

As a user, I want to log out securely

BOOK-22: Create Logout Endpoint 🔴 Critical | Size: S (2-3h)

Description:
Allow users to log out and invalidate tokens

Acceptance Criteria:
- [ ] POST /api/auth/logout endpoint created
- [ ] Requires authentication
- [ ] Removes refresh token from database
- [ ] Clears refresh token cookie
- [ ] Returns success message

Technical Notes:
- Access tokens can't be invalidated (stateless)
- They expire naturally after 15 minutes
- Focus on invalidating refresh token

Files to Update:
- controllers/authController.js
- services/authService.js

Story 5: Role-Based Access Control (RBAC)

As an admin, I want different permissions for different user roles

BOOK-25: Create Authorization Middleware 🔴 Critical | Size: M (4-5h)

Description:
Middleware to check user roles and permissions

Acceptance Criteria:
- [ ] Create authorize() middleware
- [ ] Accepts allowed roles as parameter
- [ ] Checks if req.user.role is in allowed roles
- [ ] Returns 403 if unauthorized
- [ ] Allows request if authorized

Technical Notes:
- Must run AFTER authenticate middleware
- Support multiple roles per route

Files to Create:
- middleware/authorize.middleware.js

Usage Example:
router.delete('/books/:id', authenticate, authorize(['admin', 'author']), deleteBook);

BOOK-26: Add Role Field to User Model 🔴 Critical | Size: XS (1h)

Description:
Update User model to support roles

Acceptance Criteria:
- [ ] Add role field (enum: ['user', 'author', 'admin'])
- [ ] Default role: 'user'
- [ ] Add authorInfo subdocument (for authors)
- [ ] authorInfo.isApproved (default: false)

Files to Update:
- models/User.js

Story 6: User Profile Management

As a user, I want to view and update my profile

BOOK-30: Create Get Profile Endpoint 🟠 High | Size: S (2-3h)

Description:
Get current logged-in user's profile

Acceptance Criteria:
- [ ] GET /api/users/profile endpoint created
- [ ] Requires authentication
- [ ] Returns user data (no password!)
- [ ] Includes: name, email, role, profile info

Files to Create:
- controllers/userController.js
- services/userService.js
- routes/userRoutes.js

BOOK-31: Create Update Profile Endpoint 🟠 High | Size: M (4-5h)

Description:
Allow users to update their profile

Acceptance Criteria:
- [ ] PUT /api/users/profile endpoint created
- [ ] Requires authentication
- [ ] Can update: firstName, lastName, bio, phone
- [ ] Cannot update: email, password, role (separate endpoints)
- [ ] Validates input
- [ ] Returns updated user data

Files to Update:
- controllers/userController.js
- services/userService.js

BOOK-32: Create Change Password Endpoint 🟠 High | Size: S (3-4h)

Description:
Allow users to change their password

Acceptance Criteria:
- [ ] PUT /api/users/change-password endpoint created
- [ ] Requires authentication
- [ ] Requires current password
- [ ] Validates new password strength
- [ ] Hashes new password
- [ ] Invalidates all refresh tokens (force re-login)

Files to Update:
- controllers/userController.js
- services/userService.js

📝 Phase 1 Summary Checklist

Copy this into your Jira Epic:

User Management - Tasks Checklist:

Setup & Models:
[ ] BOOK-10: Setup User Model
[ ] BOOK-26: Add Role Field to User Model

Registration:
[ ] BOOK-11: Create User Registration Endpoint
[ ] BOOK-12: Add Email Verification (optional)

Login & Auth:
[ ] BOOK-15: Setup JWT Configuration
[ ] BOOK-16: Create Login Endpoint
[ ] BOOK-17: Create Authentication Middleware

Token Management:
[ ] BOOK-20: Create Refresh Token Endpoint
[ ] BOOK-22: Create Logout Endpoint

Authorization:
[ ] BOOK-25: Create Authorization Middleware

Profile Management:
[ ] BOOK-30: Create Get Profile Endpoint
[ ] BOOK-31: Create Update Profile Endpoint
[ ] BOOK-32: Create Change Password Endpoint

Testing:
[ ] Test all endpoints with Postman
[ ] Write unit tests for services
[ ] Document API endpoints

🚀 PHASE 2: BOOK MANAGEMENT (AFTER PHASE 1)

Epic: Book CRUD Operations

Story 1: Book Model Setup

As a system, I need to store book information

BOOK-50: Create Book Model 🔴 Critical | Size: M (4-6h)

Description:
Create MongoDB schema for books

Acceptance Criteria:
- [ ] Book schema created with all fields from SRS
- [ ] Fields: isbn, title, description, author (ref to User)
- [ ] Pricing for digital and physical formats
- [ ] Metadata: genre, tags, language, publisher, etc.
- [ ] Status field (draft, pending, approved, rejected)
- [ ] Stats: totalSales, averageRating, reviewCount
- [ ] Timestamps

Files to Create:
- models/Book.js

Technical Notes:
- Reference User model for author
- Use subdocuments for pricing, metadata, stats
- Add indexes on title, isbn, genre (for search)

Story 2: Book CRUD for Authors

As an author, I want to create and manage my books

BOOK-51: Create Book (Authors) 🔴 Critical | Size: M (5-7h)

Description:
Allow authors to create new books

Acceptance Criteria:
- [ ] POST /api/books endpoint created
- [ ] Requires authentication
- [ ] Only authors can create books
- [ ] Validates all required fields
- [ ] Sets author from logged-in user
- [ ] Sets status to 'draft'
- [ ] Returns created book

Files to Create:
- controllers/bookController.js
- services/bookService.js
- routes/bookRoutes.js
- middleware/validation/bookValidation.js

Protection:
- authenticate middleware
- authorize(['author', 'admin'])

BOOK-52: Get My Books (Authors) 🔴 Critical | Size: S (2-3h)

Description:
Authors can view their own books

Acceptance Criteria:
- [ ] GET /api/author/books endpoint created
- [ ] Returns only logged-in author's books
- [ ] Filter by status (optional query param)
- [ ] Paginated results

Files to Update:
- controllers/bookController.js
- services/bookService.js

BOOK-53: Update Book (Authors) 🔴 Critical | Size: M (4-5h)

Description:
Authors can update their book metadata

Acceptance Criteria:
- [ ] PUT /api/books/:id endpoint created
- [ ] Authors can only update their own books
- [ ] Can only update if status is 'draft' or 'rejected'
- [ ] Cannot update if status is 'pending' or 'approved'
- [ ] Validates input

Files to Update:
- controllers/bookController.js
- services/bookService.js

BOOK-54: Delete Book (Authors) 🔴 Critical | Size: S (2-3h)

Description:
Authors can delete unpublished books

Acceptance Criteria:
- [ ] DELETE /api/books/:id endpoint created
- [ ] Authors can only delete their own books
- [ ] Can only delete if status is 'draft' or 'rejected'
- [ ] Returns 403 if trying to delete published book

Files to Update:
- controllers/bookController.js
- services/bookService.js

Story 3: Public Book Browsing

As a user, I want to browse and view books

BOOK-60: Get All Books (Public) 🔴 Critical | Size: M (5-6h)

Description:
Public endpoint to browse all approved books

Acceptance Criteria:
- [ ] GET /api/books endpoint created
- [ ] No authentication required
- [ ] Only returns approved books
- [ ] Supports pagination (page, limit)
- [ ] Supports filtering (genre, price range, format)
- [ ] Supports sorting (price, rating, newest)
- [ ] Returns book list + pagination info

Query Params:
- page (default: 1)
- limit (default: 20)
- genre (optional)
- minPrice, maxPrice (optional)
- format (digital/physical)
- sort (price_asc, price_desc, rating, newest)

Files to Update:
- controllers/bookController.js
- services/bookService.js

BOOK-61: Get Single Book (Public) 🔴 Critical | Size: S (2-3h)

Description:
View detailed information for a single book

Acceptance Criteria:
- [ ] GET /api/books/:id endpoint created
- [ ] No authentication required
- [ ] Only returns approved books
- [ ] Returns full book details
- [ ] Returns 404 if book not found or not approved
- [ ] Increment viewCount

Files to Update:
- controllers/bookController.js
- services/bookService.js

Story 4: Admin Book Moderation

As an admin, I want to review and approve books

BOOK-70: Get Pending Books (Admin) 🟠 High | Size: S (2-3h)

Description:
View all books pending approval

Acceptance Criteria:
- [ ] GET /api/admin/books/pending endpoint created
- [ ] Requires admin role
- [ ] Returns books with status 'pending'
- [ ] Paginated results

Files to Create:
- controllers/adminController.js
- routes/adminRoutes.js

Protection:
- authenticate
- authorize(['admin'])

BOOK-71: Approve Book (Admin) 🟠 High | Size: S (3-4h)

Description:
Admin can approve a book for publication

Acceptance Criteria:
- [ ] PUT /api/admin/books/:id/approve endpoint created
- [ ] Requires admin role
- [ ] Changes status from 'pending' to 'approved'
- [ ] Sets publishedAt timestamp
- [ ] Records admin who approved
- [ ] Sends email notification to author

Files to Update:
- controllers/adminController.js
- services/bookService.js

BOOK-72: Reject Book (Admin) 🟠 High | Size: S (3-4h)

Description:
Admin can reject a book with reason

Acceptance Criteria:
- [ ] PUT /api/admin/books/:id/reject endpoint created
- [ ] Requires admin role
- [ ] Requires rejection reason
- [ ] Changes status from 'pending' to 'rejected'
- [ ] Saves rejection reason
- [ ] Records admin who rejected
- [ ] Sends email notification to author with reason

Files to Update:
- controllers/adminController.js
- services/bookService.js

📝 Phase 2 Summary Checklist

Book Management - Tasks Checklist:

Model:
[ ] BOOK-50: Create Book Model

Author CRUD:
[ ] BOOK-51: Create Book (Authors)
[ ] BOOK-52: Get My Books (Authors)
[ ] BOOK-53: Update Book (Authors)
[ ] BOOK-54: Delete Book (Authors)

Public Browsing:
[ ] BOOK-60: Get All Books (Public)
[ ] BOOK-61: Get Single Book (Public)

Admin Moderation:
[ ] BOOK-70: Get Pending Books (Admin)
[ ] BOOK-71: Approve Book (Admin)
[ ] BOOK-72: Reject Book (Admin)

Testing:
[ ] Test all CRUD operations

[ ] Test role-based permissions
[ ] Test pagination and filters

🚀 PHASE 3: FILE UPLOAD & CLOUDINARY (AFTER PHASE 2)

Epic: PDF Upload & Storage

BOOK-100: Setup Cloudinary Account 🔴 Critical | Size: XS (1h)

Description:
Setup Cloudinary for PDF storage

Acceptance Criteria:
- [ ] Create Cloudinary account
- [ ] Get API credentials (cloud_name, api_key, api_secret)
- [ ] Add credentials to .env
- [ ] Install cloudinary SDK

Dependencies:
- npm install cloudinary multer multer-storage-cloudinary

BOOK-101: Configure Cloudinary 🔴 Critical | Size: S (2-3h)

Description:
Setup Cloudinary configuration in project

Acceptance Criteria:
- [ ] Create Cloudinary config file
- [ ] Configure upload settings
- [ ] Setup folder structure (books, covers, previews)
- [ ] Configure allowed file types (PDF only)
- [ ] Set max file size (100MB)

Files to Create:
- config/cloudinary.config.js
- utils/uploadUtils.js

BOOK-102: Create PDF Upload Endpoint 🔴 Critical | Size: L (1-2 days)

Description:
Allow authors to upload PDF files

Acceptance Criteria:
- [ ] POST /api/books/:id/upload-pdf endpoint created
- [ ] Requires authentication (author only)
- [ ] Accepts PDF file via multipart/form-data
- [ ] Validates file type (PDF only)
- [ ] Validates file size (max 100MB)
- [ ] Uploads to Cloudinary
- [ ] Stores Cloudinary URL in book record
- [ ] Returns upload progress (if possible)
- [ ] Handles upload errors

Technical Notes:
- Use multer for file handling
- Consider chunked upload for large files
- Show progress to user

Files to Create:
- middleware/upload.middleware.js

Files to Update:
- controllers/bookController.js
- services/bookService.js

BOOK-103: Create Cover Image Upload 🟠 **High** | Size: M (4-5h)

Description:
Allow authors to upload book cover images

Acceptance Criteria:
- [ ] POST /api/books/:id/upload-cover endpoint created
- [ ] Accepts image files (JPG, PNG)
- [ ] Validates file size (max 5MB)
- [ ] Uploads to Cloudinary
- [ ] Auto-resize to optimal dimensions
- [ ] Stores Cloudinary URL in book record

Files to Update:
- controllers/bookController.js
- services/bookService.js

BOOK-104: Generate Preview 🟡 Medium | Size: L (1-2 days)

Description:
Generate preview (first 5-10 pages) from uploaded PDF

Acceptance Criteria:
- [ ] After PDF upload, extract first 10 pages
- [ ] Store preview separately in Cloudinary
- [ ] Update book record with preview URL
- [ ] Make preview publicly accessible

Technical Notes:
- May need pdf-lib or similar library
- Consider using Cloudinary transformations
- This is complex, can be skipped initially

Dependencies:
- npm install pdf-lib (or similar)

🚀 PHASE 4: E-COMMERCE FEATURES

Epic: Shopping Cart

BOOK-150: Create Cart Model 🔴 Critical | Size: S (2-3h)

Description:
Create MongoDB schema for shopping carts

Acceptance Criteria:
- [ ] Cart schema created
- [ ] One cart per user
- [ ] Items array with: bookId, format, quantity, price
- [ ] Timestamps

Files to Create:
- models/Cart.js

BOOK-151: Add to Cart 🔴 Critical | Size: M (4-5h)

Description:
Allow users to add books to cart

Acceptance Criteria:
- [ ] POST /api/cart/items endpoint created
- [ ] Requires authentication
- [ ] Accepts: bookId, format (digital/physical), quantity
- [ ] Creates cart if doesn't exist
- [ ] Adds item or updates quantity if already in cart
- [ ] Validates book exists and is available
- [ ] Returns updated cart

Files to Create:
- controllers/cartController.js
- services/cartService.js
- routes/cartRoutes.js

BOOK-152: View Cart 🔴 Critical | Size: S (2-3h)

Description:
User can view their cart

Acceptance Criteria:
- [ ] GET /api/cart endpoint created
- [ ] Requires authentication
- [ ] Returns cart with populated book details
- [ ] Calculates subtotal, total
- [ ] Returns empty cart if none exists

Files to Update:
- controllers/cartController.js
- services/cartService.js

BOOK-153: Update Cart Item 🔴 Critical | Size: S (2-3h)

Description:
Update quantity of item in cart

Acceptance Criteria:
- [ ] PUT /api/cart/items/:bookId endpoint created
- [ ] Requires authentication
- [ ] Updates quantity
- [ ] Removes item if quantity = 0
- [ ] Returns updated cart

Files to Update:
- controllers/cartController.js
- services/cartService.js

BOOK-154: Remove from Cart 🔴 Critical | Size: XS (1-2h)

Description:
Remove item from cart

Acceptance Criteria:
- [ ] DELETE /api/cart/items/:bookId endpoint created
- [ ] Requires authentication
- [ ] Removes specified item
- [ ] Returns updated cart

Files to Update:
- controllers/cartController.js
- services/cartService.js

Epic: Order Processing

BOOK-200: Create Order Model 🔴 Critical | Size: M (4-5h)

Description:
Create MongoDB schema for orders

Acceptance Criteria:
- [ ] Order schema created with all fields from SRS
- [ ] orderNumber (unique, auto-generated)
- [ ] userId reference
- [ ] items array
- [ ] pricing (subtotal, tax, shipping, total)
- [ ] shippingAddress (if physical books)
- [ ] payment info
- [ ] fulfillment status

Files to Create:
- models/Order.js

BOOK-201: Setup Payment Gateway (Stripe) 🔴 Critical | Size: M (5-7h)

Description:
Integrate Stripe for payments

Acceptance Criteria:
- [ ] Create Stripe account
- [ ] Get API keys (test mode)
- [ ] Install Stripe SDK
- [ ] Add keys to .env
- [ ] Create Stripe config file

Dependencies:
- npm install stripe

Files to Create:
- config/stripe.config.js
- services/paymentService.js

BOOK-202: Create Checkout Endpoint 🔴 Critical | Size: L (1-2 days)

Description:
Process checkout and create order

Acceptance Criteria:
- [ ] POST /api/orders/checkout endpoint created
- [ ] Requires authentication
- [ ] Reads user's cart
- [ ] Validates cart not empty
- [ ] Collects shipping address (if physical books)
- [ ] Processes payment via Stripe
- [ ] Creates order record
- [ ] Clears cart after successful order
- [ ] Generates order number
- [ ] Sends confirmation email
- [ ] Returns order details

Technical Notes:
- Use transaction for atomicity
- Handle payment failures gracefully
- Store payment transaction ID

Files to Update:
- controllers/orderController.js
- services/orderService.js
- routes/orderRoutes.js

BOOK-203: View Order History 🟠 High | Size: S (2-3h)

Description:
User can view their past orders

Acceptance Criteria:
- [ ] GET /api/orders endpoint created
- [ ] Requires authentication
- [ ] Returns user's orders (paginated)
- [ ] Sorted by date (newest first)

Files to Update:
- controllers/orderController.js
- services/orderService.js

BOOK-204: View Order Details 🟠 High | Size: S (2-3h)

Description:
View single order details

Acceptance Criteria:
- [ ] GET /api/orders/:id endpoint created
- [ ] Requires authentication
- [ ] User can only view their own orders
- [ ] Returns full order details

Files to Update:
- controllers/orderController.js
- services/orderService.js

BOOK-205: Generate Invoice 🟡 Medium | Size: M (5-7h)

Description:
Generate PDF invoice for orders

Acceptance Criteria:
- [ ] Generate invoice PDF after order creation
- [ ] Store invoice URL in order record
- [ ] Include: order details, items, pricing, user info
- [ ] GET /api/orders/:id/invoice endpoint for download

Technical Notes:
- Use pdfkit or similar library
- Store invoices in Cloudinary

Dependencies:
- npm install pdfkit

🚀 PHASE 5: DIGITAL LIBRARY & READER

Epic: User Library

BOOK-250: Create ReadingProgress Model 🔴 Critical | Size: S (2-3h)

Description:
Track user reading progress

Acceptance Criteria:
- [ ] ReadingProgress schema created
- [ ] Fields: userId, bookId, currentPage, totalPages, percentage
- [ ] lastReadAt timestamp

Files to Create:
- models/ReadingProgress.js

BOOK-251: Get User Library 🔴 Critical | Size: M (4-5h)

Description:
Display user's purchased digital books

Acceptance Criteria:
- [ ] GET /api/library endpoint created
- [ ] Requires authentication
- [ ] Returns books from user's completed orders (digital format)
- [ ] Include reading progress for each book
- [ ] Support filtering and sorting

Files to Create:
- controllers/libraryController.js
- services/libraryService.js
- routes/libraryRoutes.js

BOOK-252: Generate Reader URL 🔴 Critical | Size: M (5-6h)

Description:
Generate signed URL for reading books

Acceptance Criteria:
- [ ] GET /api/library/books/:bookId/reader endpoint created
- [ ] Requires authentication
- [ ] Verifies user owns the book (purchased)
- [ ] Generates Cloudinary signed URL (30-min expiry)
- [ ] Returns URL + current progress

Technical Notes:
- Use Cloudinary's signed URL feature
- Set expiration time
- Don't allow direct PDF download

Files to Update:
- controllers/libraryController.js
- services/libraryService.js

BOOK-253: Save Reading Progress 🔴 Critical | Size: S (3-4h)

Description:
Save user's current page

Acceptance Criteria:
- [ ] PUT /api/library/books/:bookId/progress endpoint created
- [ ] Requires authentication
- [ ] Accepts: currentPage
- [ ] Calculates progress percentage
- [ ] Updates lastReadAt timestamp
- [ ] Returns updated progress

Files to Update:
- controllers/libraryController.js
- services/libraryService.js

🚀 PHASE 6: SOCIAL FEATURES

Epic: Reviews & Ratings

BOOK-300: Create Review Model 🔴 Critical | Size: S (2-3h)

Description:
Create MongoDB schema for reviews

Acceptance Criteria:
- [ ] Review schema created
- [ ] Fields: bookId, userId, rating (1-5), reviewText
- [ ] isVerifiedPurchase boolean
- [ ] helpfulCount, helpfulBy array
- [ ] status field

Files to Create:
- models/Review.js

BOOK-301: Create Review 🟠 High | Size: M (4-5h)

Description:
Users can leave reviews for purchased books

Acceptance Criteria:
- [ ] POST /api/books/:bookId/reviews endpoint created
- [ ] Requires authentication
- [ ] Validates user purchased the book
- [ ] Rating required (1-5 stars)
- [ ] Review text optional (max 1000 chars)
- [ ] One review per user per book
- [ ] Sets isVerifiedPurchase if user bought it
- [ ] Updates book's averageRating and reviewCount
- [ ] Returns created review

Technical Notes:
- Check Order model to verify purchase
- Use transaction to update book stats atomically

Files to Create:
- controllers/reviewController.js
- services/reviewService.js
- routes/reviewRoutes.js

Test Cases:
- User who purchased can review
- User who didn't purchase cannot review
- Cannot create duplicate review

BOOK-302: Get Book Reviews 🟠 High | Size: S (2-3h)

Description:
Display reviews for a book

Acceptance Criteria:
- [ ] GET /api/books/:bookId/reviews endpoint created
- [ ] No authentication required
- [ ] Returns paginated reviews
- [ ] Sort options: helpful, recent, rating
- [ ] Filter by rating (5-star, 4-star, etc.)
- [ ] Include reviewer name (not email)
- [ ] Show verified purchase badge

Query Params:
- page (default: 1)
- limit (default: 10)
- sort (helpful, recent, rating)
- rating (optional filter)

Files to Update:
- controllers/reviewController.js
- services/reviewService.js

BOOK-303: Update Own Review 🟡 Medium | Size: S (2-3h)

Description:
Users can edit their own reviews

Acceptance Criteria:
- [ ] PUT /api/reviews/:id endpoint created
- [ ] Requires authentication
- [ ] User can only update their own review
- [ ] Can update rating and/or text
- [ ] Updates book's averageRating
- [ ] Returns updated review

Files to Update:
- controllers/reviewController.js
- services/reviewService.js

BOOK-304: Delete Review 🟡 Medium | Size: S (2-3h)

Description:
Users/admins can delete reviews

Acceptance Criteria:
- [ ] DELETE /api/reviews/:id endpoint created
- [ ] Requires authentication
- [ ] Users can delete their own reviews
- [ ] Admins can delete any review
- [ ] Updates book's averageRating and reviewCount
- [ ] Returns success message

Files to Update:
- controllers/reviewController.js
- services/reviewService.js

BOOK-305: Mark Review as Helpful 🟡 Medium | Size: S (2-3h)

Description:
Users can mark reviews as helpful

Acceptance Criteria:
- [ ] POST /api/reviews/:id/helpful endpoint created
- [ ] Requires authentication
- [ ] Adds user to helpfulBy array
- [ ] Increments helpfulCount
- [ ] Cannot mark same review helpful twice
- [ ] Can unmark (toggle)

Files to Update:
- controllers/reviewController.js
- services/reviewService.js

Epic: Wishlist

BOOK-350: Create Wishlist Model 🔴 Critical | Size: S (2-3h)

Description:
Create MongoDB schema for wishlists

Acceptance Criteria:
- [ ] Wishlist schema created
- [ ] One wishlist per user
- [ ] Books array with: bookId, addedAt
- [ ] Timestamps

Files to Create:
- models/Wishlist.js

BOOK-351: Get User Wishlist 🔴 High | Size: S (2-3h)

Description:
View user's wishlist

Acceptance Criteria:
- [ ] GET /api/wishlist endpoint created
- [ ] Requires authentication
- [ ] Returns user's wishlisted books
- [ ] Populate book details (title, cover, price)
- [ ] Returns empty array if no wishlist

Files to Create:
- controllers/wishlistController.js
- services/wishlistService.js
- routes/wishlistRoutes.js

BOOK-352: Add to Wishlist 🔴 High | Size: S (2-3h)

Description:
Add book to wishlist

Acceptance Criteria:
- [ ] POST /api/wishlist/items endpoint created
- [ ] Requires authentication
- [ ] Accepts bookId
- [ ] Creates wishlist if doesn't exist
- [ ] Adds book if not already in wishlist
- [ ] Returns 400 if book already in wishlist
- [ ] Returns updated wishlist

Files to Update:
- controllers/wishlistController.js
- services/wishlistService.js

BOOK-353: Remove from Wishlist 🟠 High | Size: S (2-3h)

Description:
Remove book from wishlist

Acceptance Criteria:
- [ ] DELETE /api/wishlist/items/:bookId endpoint created
- [ ] Requires authentication
- [ ] Removes specified book
- [ ] Returns updated wishlist
- [ ] Returns 404 if book not in wishlist

Files to Update:
- controllers/wishlistController.js
- services/wishlistService.js

BOOK-354: Check if Book in Wishlist 🟡 Medium | Size: XS (1-2h)

Description:
Quick check if book is wishlisted

Acceptance Criteria:
- [ ] GET /api/wishlist/check/:bookId endpoint created
- [ ] Requires authentication
- [ ] Returns boolean: { inWishlist: true/false }
- [ ] Used for showing wishlist button state on UI

Files to Update:
- controllers/wishlistController.js
- services/wishlistService.js

Epic: Search & Recommendations

BOOK-400: Setup MongoDB Text Search 🔴 Critical | Size: S (2-3h)

Description:
Configure full-text search indexes

Acceptance Criteria:
- [ ] Create text index on Book model
- [ ] Index fields: title, description, genre, tags
- [ ] Set weights (title: 10, description: 5, genre: 3, tags: 1)
- [ ] Test index creation

Technical Notes:
- Run in MongoDB:

```
db.books.createIndex({
  title: "text",
  description: "text",
  genre: "text",
  tags: "text"
}, {
  weights: { title: 10, description: 5, genre: 3, tags: 1 }
})
```

Files to Update:
- models/Book.js (add indexes)

BOOK-401: Create Search Endpoint 🔴 Critical | Size: M (5-6h)

Description:
Full-text search for books

Acceptance Criteria:
- [ ] GET /api/search endpoint created
- [ ] No authentication required
- [ ] Search query parameter required (q)
- [ ] Uses MongoDB $text search
- [ ] Supports filters: genre, price range, format, rating
- [ ] Supports sorting: relevance, price, rating, newest
- [ ] Returns paginated results
- [ ] Returns search metadata (total results, query)

Query Parameters:
- q (search query, required)
- genre (optional)
- minPrice, maxPrice (optional)
- format (digital/physical, optional)
- minRating (optional)
- sort (relevance, price_asc, price_desc, rating, newest)
- page, limit

Response Format:
```
{
  "success": true,
  "data": {
    "query": "javascript",
    "results": [...],
    "pagination": {
      "currentPage": 1,
      "totalPages": 5,
      "totalResults": 95,
      "limit": 20
    }
  }
}
```

Files to Create:
- controllers/searchController.js
- services/searchService.js
- routes/searchRoutes.js

Technical Notes:

- Use MongoDB $text and $search operators
- Calculate text score for relevance sorting
- Apply filters after text search

BOOK-402: Create Autocomplete Endpoint 🟠 High | Size: M (4-5h)

Description:
Search suggestions as user types

Acceptance Criteria:
- [ ] GET /api/search/autocomplete endpoint created
- [ ] No authentication required
- [ ] Accepts partial query (min 2 characters)
- [ ] Returns top 10 matching books
- [ ] Returns: title, author, cover (thumbnail)
- [ ] Fast response (< 100ms)
- [ ] Uses regex or text search

Query Parameters:
- q (partial query, min 2 chars)

Response Format:

```
{
 "success": true,
 "data": {
  "suggestions": [
   {
    "id": "...",
    "title": "JavaScript: The Good Parts",
    "author": "Douglas Crockford",
    "coverUrl": "..."
   }
  ]
 }
}
```

Files to Update:
- controllers/searchController.js
- services/searchService.js

Technical Notes:
- Consider caching popular searches
- Limit fields returned for performance
- Use regex for prefix matching

BOOK-403: Get Book Recommendations 🟡 Medium | Size: M (5-7h)

Description:
Show recommended books based on current book

Acceptance Criteria:
- [ ] GET /api/books/:bookId/recommendations endpoint created
- [ ] No authentication required
- [ ] Returns 5-10 similar books
- [ ] Based on: same genre, same author, similar tags
- [ ] Exclude current book
- [ ] Prioritize books with good ratings

Algorithm (Simple Rule-Based):
1. Same author (if any)
2. Same primary genre
3. Overlapping tags
4. Sort by rating/popularity

Files to Create:
- services/recommendationService.js

Files to Update:
- controllers/bookController.js

Technical Notes:
- Start simple, can enhance with ML later
- Consider user's purchase history (future)

BOOK-404: Get Personalized Recommendations 🟡 Medium | Size: L (1-2 days)

Description:
Homepage recommendations for logged-in users

Acceptance Criteria:
- [ ] GET /api/recommendations endpoint created
- [ ] Requires authentication
- [ ] Based on user's purchase history
- [ ] Based on user's wishlist
- [ ] Based on user's browsing (if tracked)
- [ ] Returns 10-20 books
- [ ] Exclude already purchased books

Algorithm (Simple):
1. Get user's purchased genres
2. Get user's wishlisted genres
3. Find popular books in those genres
4. Filter out already owned
5. Sort by rating/sales

Files to Update:
- services/recommendationService.js
- controllers/searchController.js

Technical Notes:
- Can be enhanced with collaborative filtering later
- Consider caching per user (refresh daily)

BOOK-405: Track Search Analytics 🟢 Low | Size: M (4-5h)

Description:
Log search queries for analytics (optional)

Acceptance Criteria:
- [ ] Create SearchLog model
- [ ] Log: query, userId (if logged in), resultCount, timestamp
- [ ] Track popular searches
- [ ] Admin can view search trends

Files to Create:
- models/SearchLog.js

Files to Update:
- services/searchService.js

Note: Optional feature, can be done later

🚀 PHASE 7: DASHBOARDS & ANALYTICS

Epic: Author Dashboard

BOOK-450: Get Author Sales Stats 🟠 High | Size: M (5-6h)

Description:
Show author's sales statistics

Acceptance Criteria:
- [ ] GET /api/author/sales endpoint created
- [ ] Requires authentication (author role)
- [ ] Returns total sales count
- [ ] Returns total revenue
- [ ] Breakdown by book
- [ ] Optional date range filter (startDate, endDate)

Response Format:

```
{
 "success": true,
 "data": {
  "totalSales": 1250,
  "totalRevenue": 12450.00,
  "bookStats": [
   {
    "bookId": "...",
    "title": "...",
    "salesCount": 350,
    "revenue": 3500.00
   }
  ]
 }
}
```

Files to Create:
- controllers/authorController.js
- services/authorService.js
- routes/authorRoutes.js

Technical Notes:
- Query Order model for completed orders
- Aggregate by bookId
- Filter by author's books only

BOOK-451: Get Sales Chart Data 🟡 Medium | Size: M (4-5h)

Description:
Sales over time for charts

Acceptance Criteria:
- [ ] GET /api/author/sales/chart endpoint created
- [ ] Requires authentication (author role)
- [ ] Returns daily/weekly/monthly sales
- [ ] Date range parameter (last 7 days, 30 days, 90 days, 1 year)
- [ ] Format suitable for chart libraries

Response Format:

```
{
 "success": true,
 "data": {
  "chartData": [
   { "date": "2025-12-01", "sales": 15, "revenue": 150.00 },
   { "date": "2025-12-02", "sales": 23, "revenue": 230.00 }
  ]
 }
}
```

Files to Update:
- controllers/authorController.js
- services/authorService.js

BOOK-452: Export Sales Data 🟡 Medium | Size: M (4-5h)

Description:
Export sales data as CSV

Acceptance Criteria:
- [ ] GET /api/author/sales/export endpoint created
- [ ] Requires authentication (author role)
- [ ] Returns CSV file
- [ ] Includes: date, book title, quantity, revenue
- [ ] Optional date range filter

Files to Update:
- controllers/authorController.js
- services/authorService.js

Technical Notes:
- Use csv-writer or similar library
- Set proper headers for file download

Dependencies:
- npm install csv-writer

Epic: Admin Dashboard

BOOK-500: Get Admin Dashboard Stats 🟠 High | Size: M (5-6h)

Description:
Overview statistics for admin

Acceptance Criteria:
- [ ] GET /api/admin/dashboard endpoint created
- [ ] Requires authentication (admin role)
- [ ] Returns key metrics:
  - Total users (by role)
  - Total books (by status)
  - Pending book approvals count
  - Total orders
  - Today's revenue
  - This month's revenue
- [ ] Returns recent activity

Response Format:
```
{
  "success": true,
  "data": {
    "users": {
      "total": 5420,
      "authors": 234,
      "admins": 5
    },
    "books": {
      "total": 1850,
      "pending": 23,
      "approved": 1750,
      "rejected": 77
    },
    "orders": {
      "total": 8950,
      "today": 45
    },
    "revenue": {
      "today": 1250.00,
      "thisMonth": 45600.00
    }
  }
}
```

Files to Update:

- controllers/adminController.js
- services/adminService.js

BOOK-501: Get User Management List 🔴 High | Size: M (4-5h)

Description:
List all users for management

Acceptance Criteria:
- [ ] GET /api/admin/users endpoint created
- [ ] Requires authentication (admin role)
- [ ] Returns paginated user list
- [ ] Filter by role
- [ ] Filter by status (active/inactive)
- [ ] Search by name or email
- [ ] Sort options

Files to Update:
- controllers/adminController.js
- services/adminService.js

BOOK-502: Update User Role 🔴 High | Size: S (3-4h)

Description:
Admin can change user roles

Acceptance Criteria:
- [ ] PUT /api/admin/users/:id/role endpoint created
- [ ] Requires authentication (admin role)
- [ ] Can change role (user, author, admin)
- [ ] Validates new role
- [ ] Cannot demote yourself (admin safety)
- [ ] Logs role change

Files to Update:
- controllers/adminController.js
- services/adminService.js

BOOK-503: Deactivate/Activate User 🔴 High | Size: S (2-3h)

Description:
Admin can deactivate user accounts

Acceptance Criteria:
- [ ] PUT /api/admin/users/:id/status endpoint created
- [ ] Requires authentication (admin role)
- [ ] Toggle isActive status
- [ ] Cannot deactivate yourself
- [ ] Invalidate user's refresh tokens if deactivated
- [ ] Logs status change

Files to Update:
- controllers/adminController.js
- services/adminService.js

BOOK-504: Get All Orders (Admin) 🟠 High | Size: M (4-5h)

Description:
Admin can view all orders

Acceptance Criteria:
- [ ] GET /api/admin/orders endpoint created
- [ ] Requires authentication (admin role)
- [ ] Returns paginated orders
- [ ] Filter by status
- [ ] Filter by date range
- [ ] Search by order number or user email
- [ ] Sort options

Files to Update:
- controllers/adminController.js
- services/adminService.js

BOOK-505: Update Order Status 🟠 High | Size: S (3-4h)

Description:
Admin can update order fulfillment status

Acceptance Criteria:
- [ ] PUT /api/admin/orders/:id/status endpoint created
- [ ] Requires authentication (admin role)
- [ ] Can update fulfillment status
- [ ] Status flow: pending → processing → shipped → delivered
- [ ] Optional: add tracking number
- [ ] Send email notification to customer
- [ ] Update timestamps (shippedAt, deliveredAt)

Files to Update:
- controllers/adminController.js
- services/adminService.js

BOOK-506: Generate Platform Reports 🟡 Medium | Size: L (1-2 days)

Description:
Generate various reports for analysis

Acceptance Criteria:
- [ ] POST /api/admin/reports endpoint created
- [ ] Requires authentication (admin role)
- [ ] Report types:
  - Sales report (by date range)
  - User registration report
  - Revenue by genre/author
  - Top selling books
- [ ] Export as PDF or CSV
- [ ] Email report option

Files to Create:
- services/reportService.js

Files to Update:
- controllers/adminController.js

Technical Notes:
- Use aggregation pipelines
- Consider background jobs for large reports

Dependencies:
- npm install pdfkit csv-writer

📋 PHASE 6 & 7 SUMMARY CHECKLIST

=== REVIEWS & RATINGS ===
[ ] BOOK-300: Create Review Model
[ ] BOOK-301: Create Review
[ ] BOOK-302: Get Book Reviews
[ ] BOOK-303: Update Own Review
[ ] BOOK-304: Delete Review
[ ] BOOK-305: Mark Review as Helpful

=== WISHLIST ===
[ ] BOOK-350: Create Wishlist Model
[ ] BOOK-351: Get User Wishlist

[ ] BOOK-352: Add to Wishlist
[ ] BOOK-353: Remove from Wishlist
[ ] BOOK-354: Check if Book in Wishlist

=== SEARCH & RECOMMENDATIONS ===
[ ] BOOK-400: Setup MongoDB Text Search
[ ] BOOK-401: Create Search Endpoint
[ ] BOOK-402: Create Autocomplete Endpoint
[ ] BOOK-403: Get Book Recommendations
[ ] BOOK-404: Get Personalized Recommendations
[ ] BOOK-405: Track Search Analytics (Optional)

=== AUTHOR DASHBOARD ===
[ ] BOOK-450: Get Author Sales Stats
[ ] BOOK-451: Get Sales Chart Data
[ ] BOOK-452: Export Sales Data

=== ADMIN DASHBOARD ===
[ ] BOOK-500: Get Admin Dashboard Stats
[ ] BOOK-501: Get User Management List
[ ] BOOK-502: Update User Role
[ ] BOOK-503: Deactivate/Activate User
[ ] BOOK-504: Get All Orders (Admin)
[ ] BOOK-505: Update Order Status
[ ] BOOK-506: Generate Platform Reports

📊 COMPLETE PROJECT OVERVIEW

Total Tasks by Phase:

- ✅ Phase 0 (Category): 5 tasks - DONE!
- 🔐 Phase 1 (User Management): 13 tasks
- 📚 Phase 2 (Book Management): 10 tasks
- ☁️ Phase 3 (File Upload): 5 tasks
- 🛒 Phase 4 (E-commerce): 11 tasks
- 📖 Phase 5 (Digital Library): 4 tasks
- ⭐ Phase 6 (Social Features): 17 tasks
- 📊 Phase 7 (Dashboards): 10 tasks

TOTAL: 75 tasks (1 completed, 74 remaining)

Estimated Timeline:

- Phase 1: 2-3 weeks
- Phase 2: 1-2 weeks
- Phase 3: 1 week
- Phase 4: 2 weeks
- Phase 5: 1 week
- Phase 6: 2-3 weeks
- Phase 7: 1-2 weeks

TOTAL: 10-14 weeks (2.5-3.5 months)

🎯 JIRA IMPORT FORMAT

For CSV Import to Jira:

Create a CSV file with these columns:

```
Summary,Description,Issue Type,Priority,Story Points,Labels
```

Example rows:

```csv
"BOOK-10: Setup User Model","Create MongoDB schema for users with all necessary fields. Include: email, password, first
"BOOK-11: Create User Registration Endpoint","Build POST /api/auth/register endpoint with validation, email check, passw
```

Would you like me to generate the full CSV file for import?

✅ QUICK START GUIDE

Week 1-2: Authentication Foundation

1. Start with BOOK-10 (User Model)
2. Then BOOK-15 (JWT Setup)

3. Then BOOK-11 (Registration)
4. Then BOOK-16 (Login)
5. Then BOOK-17 (Auth Middleware)

Week 3: Complete User Management

6. BOOK-20 (Refresh Token)
7. BOOK-22 (Logout)
8. BOOK-25 (Authorization)
9. BOOK-26 (Role Field)
10. BOOK-30-32 (Profile Management)

Week 4-5: Book Management

Start Phase 2 tasks (BOOK-50 onwards)

Continue chronologically through phases

You're all set! 🚀 Let's build Bookify!