API Integration

1. Architecture

API Client Layer (lib/api/client.ts) - Provides core HTTP communication functionality

Endpoints Layer (lib/api/endpoints.ts) - Manages centralized URL definitions

Service Layer (services/books.services.ts) - Handles business logic and domain-specific error handling

2. API Client (lib/api/client.ts)

Purpose

Reusable HTTP client for all backend communication.

Key Features

- Environment-based URL configuration
- Unified error handling
- Type-safe responses
- Network failure detection (status 0)

Private Methods

handleResponse<T>(response: Response): Promise<T>

Public HTTP Methods

get<T>(endpoint: string): Promise<T>

post<T>(endpoint: string, data: unknown): Promise<T>

put<T>(endpoint: string, data: unknown): Promise<T>

delete<T>(endpoint: string): Promise<T>

3. Endpoints Management (lib/api/endpoints.ts)

Purpose

Single source of truth for all API routes.

Structure

Centralized constant object for all API endpoints.

```
export const API_ENDPOINTS = {
  BOOKS: {
    BASE: '/api/v1/books',
    BY_ID: (id: number) => `/api/v1/books/${id}`,
  },
} as const;
```

4. Books Service (services/books.services.ts)

Methods

getAll(): Promise<Book[]>

- Endpoint: GET /api/v1/books
- Returns: Array of books
- Error Handling: Returns empty array on 404
- Use Case: Fetch all books for listing pages

getById(id: number): Promise<Book | null>

- Endpoint: GET /api/v1/books/{id}
- Returns: Single book or null if not found
- Error Handling: Returns null on 404
- Use Case: Fetch single book details

create(data: BookCreateDto): Promise<Book>

- Endpoint: POST /api/v1/books
- Payload: BookCreateDto

- Returns: Created book with ID
- Error Handling: Throws on validation or server errors

update(id: number, data: BookUpdateDto): Promise<Book | null>

- Endpoint: PUT /api/v1/books/{id}
- Payload: BookUpdateDto
- Returns: Updated book or null if not found
- Error Handling: Returns null on 404

softDelete(id: number): Promise<{ message: string }>

- Endpoint: DELETE /api/v1/books/{id}
- Returns: Confirmation message
- Note: Performs soft delete (backend dependent)

UI Integration

1. Page Files (Where API Calls Happen)

Books List Page (*app/books/page.tsx*)

API Call:

- Method: *booksService.getAll()*
- Endpoint: GET */api/v1/books*
- Returns: Array of books
- Passes data to: *<BookList />* component

Book Details Page (*app/books/[id]/page.tsx*)

API Call:

- Method: *booksService.getById(id)*
- Endpoint: GET */api/v1/books/{id}*
- Returns: Single book
- Passes data to: *<BookDetails />* component

New Book Page (*app/books/new/page.tsx*)

API Call: None here. Form submits to Server Action

Edit Book Page *(app/books/[id]/edit/page.tsx*)

API Call:

- Method: *booksService.getById(id)*
- Endpoint: GET */api/v1/books/{id}*
- Returns: Book to edit
- Passes data to: *<BookForm />* component

2. Server Actions (Where Form Submissions Go)

Create Book (*app/books/new/actions.ts*)

API Call:

- Method: *booksService.create(data)*
- Endpoint: POST */api/v1/books*
- Then: Redirects to new book page

Update Book (*app/books/[id]/edit/actions.ts*)

API Call:

- Method: *booksService.update(id, data)*
- Endpoint: PUT */api/v1/books/{id}*
- Then: Redirects to updated book page

Delete Book

API Call:

- Method: *booksService.softDelete(id)*
- Endpoint: DELETE */api/v1/books/{id}*
- Then: Redirects to books list

3. Components (What Users See)

<BookList /> - Shows all books

- Receives: books[] from page
- Displays: Grid of <BookCard /> components

<BookCard /> - Shows one book preview

- Receives: book from <BookList />
- Displays: Title, author, cover
- Uses: <Card /> from ui/

<BookDetails /> - Shows full book info

- Receives: book from page
- Displays: All book information
- Has: Edit and Delete buttons

<BookForm /> - Create/edit form

- Receives: book (optional, for editing)
- Displays: Form fields using <Input /> from ui/
- Submits: To Server Action

<NewBookForm /> - Create-only form

- Receives: Nothing
- Displays: Empty form
- Submits: To createBook action

4. UI Components (ui/ folder)

<Button /> - Buttons everywhere

<Card /> - Containers for content

<Input /> - Form input fields