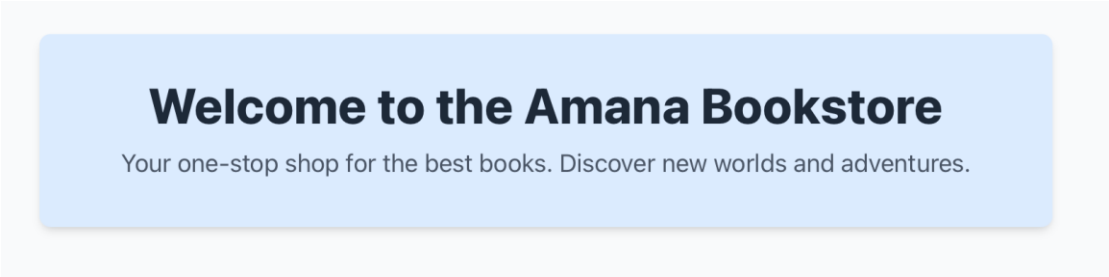
9/22/2025

Pre-Work Assignment

Layla As_Saabna

Table of Contents

Which file contains the main welcome message and hero section that users see when they first visit the website?



Answer: **page.tsx**

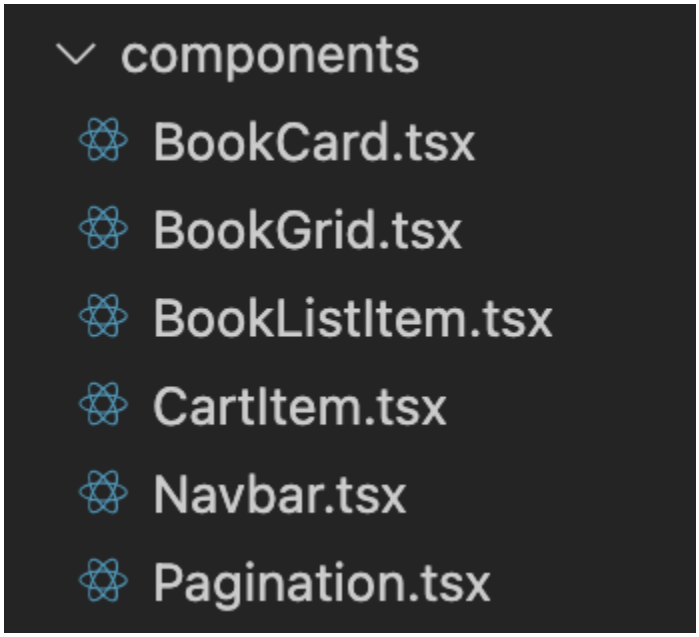
Where is all the book data stored in this application? What file would you look at to see the list of available books?

Answer: **data/books.ts**

Find the navigation bar at the top of the website. Which file controls what appears in this navigation?

Answer: **components/Navbar.tsx**

Looking inside of the components folder, what do you think each component does in the context of this application and which files reference each component?



Component Analysis for Amana Bookstore

1. Navbar.tsx

Purpose: Main navigation bar for the entire application

- Displays the "Amana Bookstore" brand/logo
- Provides navigation links (Home, My Cart)
- Shows real-time cart item count with a badge
- Manages cart state through localStorage and custom events
- Highlights the current active page

Used in:

- layout.tsx - Global layout component (appears on every page)

---

2. BookCard.tsx

Purpose: Card-style display component for individual books

- Shows book cover (placeholder icon), title, author, rating with stars
- Displays genres, price, and stock status
- Interactive "Add to Cart" button with loading states and success feedback
- "View Details" link to individual book page

- Optimized for grid layouts with hover effects

Used in:

- BookGrid.tsx - For displaying featured books in the carousel section

---

3. BookGrid.tsx

Purpose: Main catalog component that manages the entire book browsing experience

- **Featured Books Carousel**: Shows rotating featured books using BookCard components
- **Search & Filter**: Text search and genre filtering capabilities
- **Sorting**: Multiple sort options (title, author, date, rating, price)
- **Book Listing**: Shows all books in list format using BookListItem components
- **Pagination**: Implements pagination using the Pagination component
- Handles all the complex state management for browsing books

Used in:

- page.tsx - Main homepage component

---

4. BookListItem.tsx

Purpose: Horizontal list-style display for books (alternative to card view)

- More compact layout showing book info in a row format
- Same functionality as BookCard but optimized for list views
- Shows book icon, title, author, rating, genres, price
- "Add to Cart" and "View Details" buttons
- Better for displaying many books in a compact space

Used in:

- BookGrid.tsx - For the "All Books" section in list format

---

5. CartItem.tsx

Purpose: Individual cart item display component

- Shows book details within the shopping cart
- Quantity controls (+ and - buttons)
- Displays subtotal calculation for each item
- "Remove" button to delete items from cart
- Links to book detail page

Used in:

- page.tsx - Shopping cart page to display each cart item

---

6. Pagination.tsx

Purpose: Reusable pagination controls

- Page navigation with previous/next buttons
- Displays page numbers with ellipsis for large page counts
- Shows current items range ("Showing X to Y of Z books")
- Items per page selector (8, 12, 16, 24 options)
- Handles page changes and items-per-page changes

Used in:

- BookGrid.tsx - For paginating the book listings

Which file controls how Books get displayed in the All Books section? Which file controls how Books get displayed in the Featured Books Section?

All Books Section

File: BookListItem.tsx

The "All Books" section uses the BookListItem component, which displays books in a horizontal list format. This component is imported and used within BookGrid.tsx in the "All Books" section:

```
<div className="space-y-3">
 {paginatedBooks.map(book => (
   <BookListItem key={book.id} book={book} onAddToCart={onAddToCart} />
 ))}
</div>
```

BookListItem provides a compact, row-based layout showing book information horizontally with the book icon on the left and details/actions on the right.

Featured Books Section

File: BookCard.tsx

The "Featured Books" section uses the BookCard component, which displays books in a card-style grid format. This component is also imported and used within BookGrid.tsx in the featured books carousel:

```
<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-8">
 {currentFeaturedBooks.map(book => (
   <BookCard key={book.id} book={book} onAddToCart={onAddToCart} />
 ))}
</div>
```

BookCard provides a vertical card layout with larger visual emphasis, perfect for showcasing featured books.

Summary

- **All Books Section**: Controlled by BookListItem.tsx (horizontal list view)
- **Featured Books Section**: Controlled by BookCard.tsx (card grid view)

- **Both sections are orchestrated by**: BookGrid.tsx which decides which component to use for each section and manages the overall layout and data flow

The choice of different components allows the application to present the same book data in two distinct visual formats - featured books get the more prominent card treatment, while the complete catalog uses a more compact list format for better browsing of large numbers of books.

How are the properties (like price, review count, and genre) related to the books being retrieved from the data? Show the snippet of code that controls the display of these properties in the Featured Book section.

Data Flow and Property Display

1. Data Structure

The book properties are defined in the Book interface in index.ts:

```
export interface Book {
 id: string;
 title: string;
 author: string;
```

```
  // ... other properties
  price: number;         // Numeric price value
  rating: number;        // Rating from 0-5
  reviewCount: number;   // Number of reviews
  genre: string[];       // Array of genre strings
  inStock: boolean;      // Stock availability
  featured: boolean;     // Whether book is featured
}
```

2. Data Source

Books are stored as an array in books.ts. Each book object contains all these properties:

```
{
  id: '1',
  title: 'Fundamentals of Classical Mechanics',
  author: 'Dr. Ahmad Al-Kindi',
  price: 89.99,            // ← Price property
  rating: 4.8,             // ← Rating property
  reviewCount: 23,         // ← Review count property
  genre: ['Physics', 'Textbook'], // ← Genre array property
  inStock: true,
  featured: true,          // ← This determines if shown in Featured section
  // ... other properties
}
```

3. Featured Books Display Code

Here are the specific code snippets from BookCard.tsx that control how these properties are displayed in the **Featured Books section**:

Review Count Display:

```
<div className="flex items-center mt-2">
 {renderStars(book.rating)}
 <span className="text-xs text-gray-500 ml-2">({book.reviewCount} reviews)</span>
</div>
```

Genre Display:

```
<div className="mt-2">
 {book.genre.slice(0, 2).map((g) => (
  <span key={g} className="inline-block bg-gray-200 rounded-full px-2 py-1 text-xs font-semibold text-gray-700 mr-2 mb-2">
   {g}
  </span>
 ))}
 {book.genre.length > 2 && (
  <span className="text-xs text-gray-500">+{book.genre.length - 2} more</span>
 )}
</div>
```

Price Display:

```
<div className="flex items-center justify-between mt-3">
 <p className="text-xl font-bold text-gray-900">${book.price.toFixed(2)}</p>
 {!book.inStock && (
  <span className="text-xs text-red-600 font-medium">Out of Stock</span>
 )}
</div>
```

Rating Display (Stars):

The rating is displayed using the renderStars() function which converts the numeric rating (e.g., 4.8) into visual star components:

```
// Called with book.rating
{renderStars(book.rating)}
```

4. How Featured Books are Selected

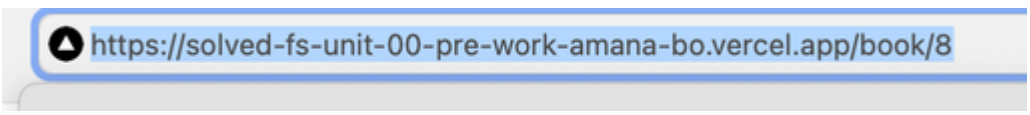
In BookGrid.tsx, featured books are filtered from the main books array:

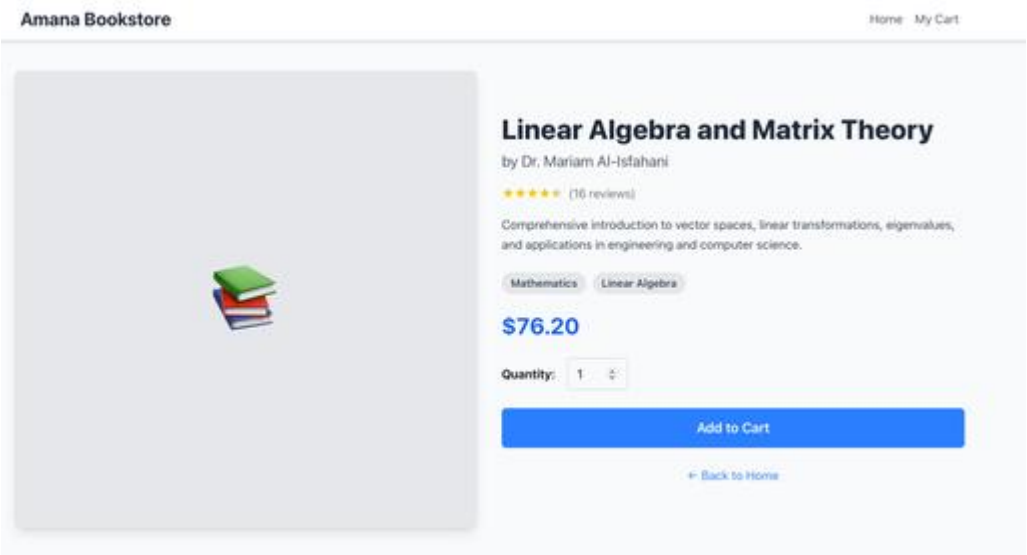
```
const featuredBooks = useMemo(() => books.filter(book => book.featured), [books]);
```

Only books with featured: true appear in the Featured Books section, and they're displayed using the BookCard component which renders all these properties as shown above.

The properties are directly accessed from the book prop passed to the BookCard component, creating a direct relationship between the data structure and the visual display.

How does this application generate unique URLs for each individual book (like /book/1, /book/2, etc.)? Which folder structure makes this possible?





Dynamic Route Structure

The unique URLs are made possible by the **folder structure** in the Next.js App Router:

src/app/

└── book/

  └── [id]/

    └── page.tsx

Key Components:

1. **[id] folder**: The square brackets [id] create a **dynamic route segment**
2. page.tsx: This file handles the rendering for any URL matching /book/[anything]

How It Works

1. URL Generation

Throughout the application, book URLs are generated using the book's id property:

In BookCard.tsx **(Featured Books):**

```
<Link href={`/book/${book.id}`} className="block cursor-pointer">
```

```
 <h3 className="text-lg font-semibold text-gray-800 truncate hover:text-blue-600 transition-colors duration-200">{book.title}</h3>
```

```
 <p className="text-sm text-gray-600 mt-1">by {book.author}</p>
</Link>
```

In BookListItem.tsx **(All Books):**

```
<Link href={`/book/${book.id}`} className="block group cursor-pointer">
 <h3 className="text-lg font-semibold text-gray-800 truncate group-hover:text-blue-600 transition-colors duration-200">
  {book.title}
 </h3>
 <p className="text-sm text-gray-600 mt-1">by {book.author}</p>
</Link>
```

In CartItem.tsx:

```
<Link href={`/book/${book.id}`} className="text-lg font-semibold text-gray-800 hover:text-blue-500 cursor-pointer">
 {book.title}
</Link>
```

2. Route Resolution

When a user visits URLs like:

- /book/1
- /book/2
- /book/physics-101

Next.js automatically:

1. Routes to page.tsx
2. Passes the URL parameter (1, 2, physics-101) as the id parameter

3. Parameter Extraction

In the page.tsx file, the dynamic id is extracted using Next.js hooks:

Here's the key code that makes this work:

```
const params = useParams();  // Next.js hook to get route parameters
const { id } = params;      // Extract the 'id' from the URL


useEffect(() => {
 if (id) {
  // Find the book with matching ID from the data
  const foundBook = books.find((b) => b.id === id);
  if (foundBook) {
   setBook(foundBook);
   // Load associated reviews
   const bookReviewsData = reviews.filter((review) => review.bookId === id);
   setBookReviews(bookReviewsData);
  } else {
   setError('Book not found.');
  }
  setIsLoading(false);
 }
}, [id]);
```

Complete Flow Example

1. **User clicks** on a book titled "Fundamentals of Classical Mechanics"
2. **Link generated**: /book/1 (where 1 is the book's ID)
3. **Next.js routes** to page.tsx
4. useParams() extracts id = "1" from the URL
5. **Data lookup**: books.find((b) => b.id === "1") finds the matching book
6. **Page renders** with that specific book's details

Folder Structure Benefits

This Next.js App Router structure provides:

- **SEO-friendly URLs**: /book/1, /book/2 are clean and indexable
- **Dynamic content**: One page component handles all book detail pages
- **Type safety**: TypeScript knows the id parameter type
- **Automatic routing**: No manual route configuration needed
- **Performance**: Only loads data for the requested book

The [id] folder naming convention is what makes this dynamic routing possible in Next.js!

Where in the codebase is the logic for keeping track of the items in the cart? Briefly, how does it work? Show screenshots of the codebase.

How Cart Management Works

1. Data Storage

- **Storage**: Uses localStorage to persist cart data across browser sessions
- **Format**: Array of CartItem objects with id, bookId, quantity, and addedAt

2. Event-Driven Synchronization

- **Custom Events**: Uses window.dispatchEvent(new CustomEvent('cartUpdated')) to notify components
- **Listeners**: Navbar listens for these events to update the cart count badge

3. Key Operations

Adding Items:

```
// Create cart item
const cartItem: CartItem = {
 id: `${book.id}-${Date.now()}`,
 bookId: book.id,
 quantity: quantity,
 addedAt: new Date().toISOString(),
};
```

```
// Get existing cart, add/update item, save back to localStorage
const cart: CartItem[] = storedCart ? JSON.parse(storedCart) : [];
// ... logic to add or update quantity
localStorage.setItem('cart', JSON.stringify(cart));

// Notify other components
window.dispatchEvent(new CustomEvent('cartUpdated'));
```

Updating Quantities:

```
const updateQuantity = (bookId: string, newQuantity: number) => {
 // Update React state
 const updatedItems = cartItems.map(item =>
   item.book.id === bookId ? { ...item, quantity: newQuantity } : item
 );
 setCartItems(updatedItems);

 // Sync to localStorage
 localStorage.setItem('cart', JSON.stringify(cartForStorage));

 // Notify navbar
 window.dispatchEvent(new CustomEvent('cartUpdated'));
};
```

Removing Items:

```
const removeItem = (bookId: string) => {
 const updatedItems = cartItems.filter(item => item.book.id !== bookId);
 setCartItems(updatedItems);
```

```
 localStorage.setItem('cart', JSON.stringify(cartForStorage));
 window.dispatchEvent(new CustomEvent('cartUpdated'));
};
```

4. Cross-Component Communication

The cart state is managed without Context API by using:

- **localStorage** for persistence
- **Custom events** for real-time updates
- **Component-specific state** for local rendering

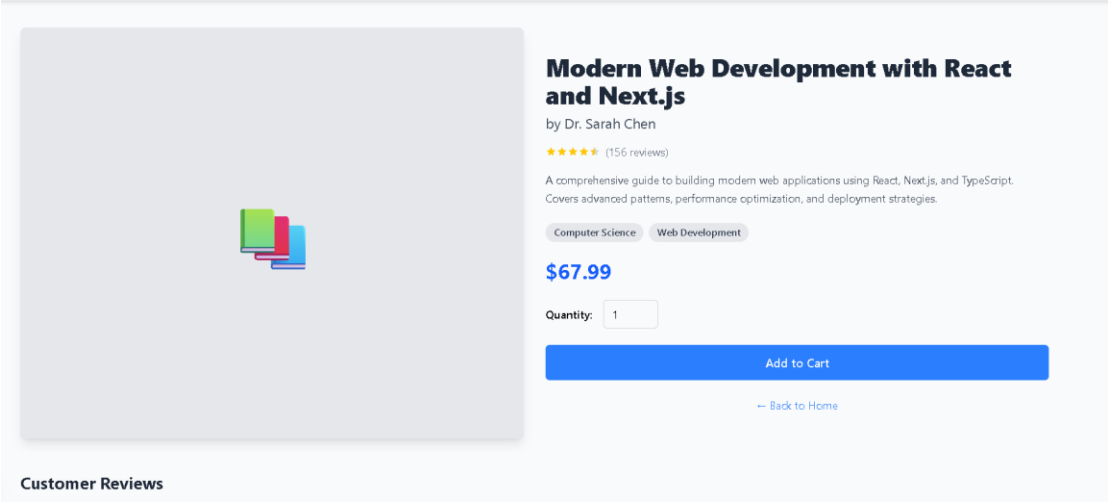
This architecture allows the cart to work across different pages while maintaining data consistency and providing real-time updates to the navigation bar.

How might we add an additional book to the collection? For this question, add a book yourself and include a screenshot showing that it worked.

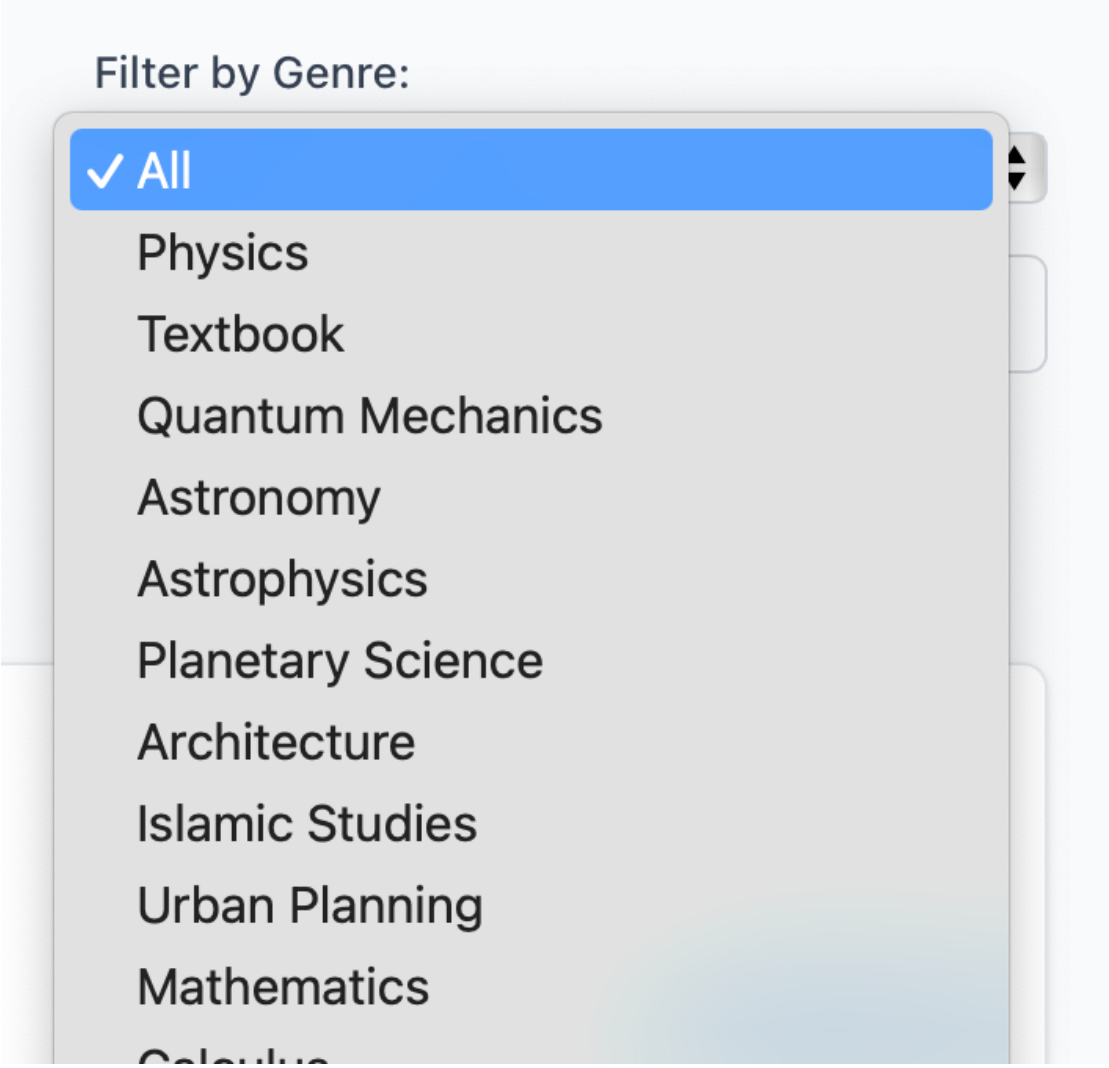
1. **Open the books data file**: books.ts
2. **Add a new book object** to the books array with all required fields:

```
{
 id: '101',                          // Unique identifier
 title: 'Modern Web Development with React and Next.js',
 author: 'Dr. Sarah Chen',
 description: 'A comprehensive guide to building modern web applications...',
 price: 67.99,                       // Price in dollars
 image: '/images/book101.jpg',       // Image path
 isbn: '978-1234567890',             // ISBN number
 genre: ['Computer Science', 'Web Development'], // Array of genres
 tags: ['React', 'Next.js', 'TypeScript', 'JavaScript'], // Keywords
 datePublished: '2024-01-15',        // Publication date
```

```
pages: 485,                    // Number of pages
language: 'English',           // Language
publisher: 'TechPress Publications',  // Publisher name
rating: 4.9,                   // Rating (0-5)
reviewCount: 156,              // Number of reviews
inStock: true,                 // Availability
featured: true,                // Show in featured section
}
```



How does the code “filter” the books that are considered the “Featured Books”? How does it “filter” books based on category? Show screenshots or code snippets from the code base.



1. Featured Books Filtering

Key Code for Featured Books Filtering:

```
// This line filters all books where the 'featured' property is true
const featuredBooks = useMemo(() => books.filter(book => book.featured), [books]);
```

This simple filter looks at each book object and includes it in the featuredBooks array only if book.featured === true.

2. Category/Genre Filtering

Key Code for Genre Filtering:

1. **Generate Genre List:**

```
// Creates an array of all unique genres from all books
const genres = useMemo(() => {
 const allGenres = books.flatMap(book => book.genre);  // Flatten all genre arrays
 return ['All', ...new Set(allGenres)];            // Remove duplicates, add 'All'
}, [books]);
```

2. **Filter Books by Genre:**

```
const filteredAndSortedBooks = useMemo(() => {
 const filtered = books.filter(book => {
  const matchesSearch =
   book.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
   book.author.toLowerCase().includes(searchQuery.toLowerCase());

  const matchesGenre =
   selectedGenre === 'All' || book.genre.includes(selectedGenre);

  return matchesSearch && matchesGenre;  // Both conditions must be true
 });
 // ... sorting logic continues
}, [books, searchQuery, selectedGenre, sortBy, sortOrder]);
```

Summary of Filtering Mechanisms

🌟 **Featured Books Filter:**

```
// Simple boolean filter
const featuredBooks = useMemo(() =>
 books.filter(book => book.featured),
 [books]
```

```
);
```

How it works:

- Checks each book's featured property
- Only includes books where featured === true
- Used for the Featured Books carousel section

🔍 **Genre/Category Filter:**

```
// 1. Extract all unique genres
const genres = useMemo(() => {
 const allGenres = books.flatMap(book => book.genre);
 return ['All', ...new Set(allGenres)];
}, [books]);


// 2. Filter books by selected genre
const matchesGenre =
 selectedGenre === 'All' || book.genre.includes(selectedGenre);
```

How it works:

1. **Extract genres**: flatMap() flattens all genre arrays from all books
2. **Remove duplicates**: new Set() creates unique list
3. **Filter logic**:
    - If "All" is selected → show all books
    - Otherwise → check if book's genre array includes the selected genre

📋 **Data Structure Supporting Filters:**

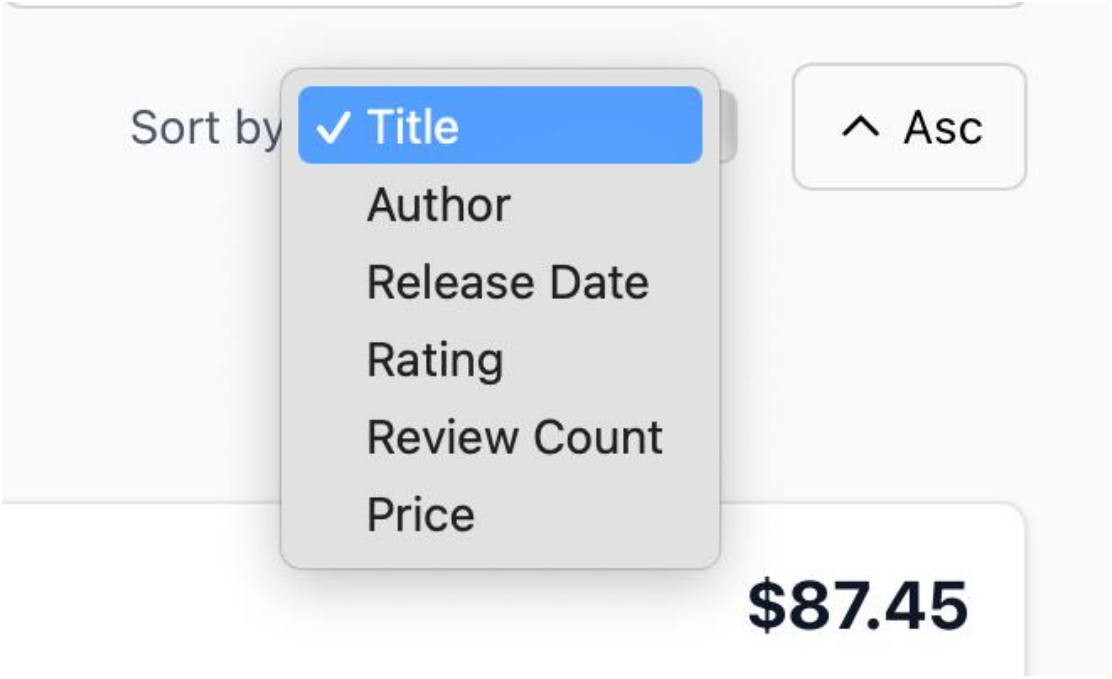
```
interface Book {
 // ... other properties
 featured: boolean;        // Used for Featured Books filter
 genre: string[];          // Array allows multiple genres per book
 // ... other properties
```

```
}
```

Example book data:

```
{
 id: '1',
 title: 'Fundamentals of Classical Mechanics',
 genre: ['Physics', 'Textbook'],  // Multiple genres possible
 featured: true,                  // Makes it appear in Featured section
 // ... other properties
}
```

How does the code “sort” books based on the user selection? Show screenshots or code snippets from the code base



📊 **How Book Sorting Works**

🎛️ **1. Sorting Controls (UI)**

The application provides two sorting controls:

```
{/* Sort By Dropdown */}
<select
 id="sortBy"
 value={sortBy}
 onChange={(e) => setSortBy(e.target.value)}
 className="px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 bg-white text-sm"
>
 <option value="title">Title</option>
 <option value="author">Author</option>
 <option value="datePublished">Release Date</option>
 <option value="rating">Rating</option>
 <option value="reviewCount">Review Count</option>
 <option value="price">Price</option>
</select>

{/* Sort Order Toggle Button */}
<button
 onClick={() => setSortOrder(sortOrder === 'asc' ? 'desc' : 'asc')}
 className="flex items-center gap-1 px-3 py-2 border border-gray-300 rounded-md hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-blue-500 text-sm cursor-pointer"
>
 {sortOrder === 'asc' ? (
  <>
   <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0 0 24 24">
```

```
    <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M5 15l7-7 7 7" />
   </svg>
   <span>Asc</span>
  </>
 ) : (
  <>
   <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0 0 24 24">
    <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M19 9l-7 7-7-7" />
   </svg>
   <span>Desc</span>
  </>
 )}
</button>
```

🔧 2. State Management

```
const [sortBy, setSortBy] = useState('title');            // What field to sort by
const [sortOrder, setSortOrder] = useState<'asc' | 'desc'>('asc'); // Direction
```

⚙️ 3. Core Sorting Algorithm

```
const filteredAndSortedBooks = useMemo(() => {
 // First filter the books (search + genre)
 const filtered = books.filter(book => {
  const matchesSearch =
   book.title.toLowerCase().includes(searchQuery.toLowerCase()) ||
   book.author.toLowerCase().includes(searchQuery.toLowerCase());
```

```
  const matchesGenre =
    selectedGenre === 'All' || book.genre.includes(selectedGenre);

  return matchesSearch && matchesGenre;
 });

 // Then sort the filtered books
 const sorted = [...filtered].sort((a, b) => {
  let comparison = 0;

  switch (sortBy) {
   case 'title':
    comparison = a.title.localeCompare(b.title);
    break;
   case 'author':
    comparison = a.author.localeCompare(b.author);
    break;
   case 'datePublished':
    comparison = new Date(a.datePublished).getTime() - new
Date(b.datePublished).getTime();
    break;
   case 'rating':
    comparison = a.rating - b.rating;
    break;
   case 'reviewCount':
    comparison = a.reviewCount - b.reviewCount;
    break;
   case 'price':
```

```
      comparison = a.price - b.price;
      break;
    default:
      comparison = 0;
  }

  return sortOrder === 'asc' ? comparison : -comparison;
 });

 return sorted;
}, [books, searchQuery, selectedGenre, sortBy, sortOrder]);
```

📋 **4. Sorting Logic Breakdown**

String Sorting (Title, Author):

```
case 'title':
 comparison = a.title.localeCompare(b.title);
 break;
case 'author':
 comparison = a.author.localeCompare(b.author);
 break;
```

- Uses localeCompare() for proper alphabetical sorting
- Handles international characters correctly

Date Sorting:

```
case 'datePublished':
 comparison = new Date(a.datePublished).getTime() - new
Date(b.datePublished).getTime();
 break;
```

- Converts date strings to Date objects

- Compares timestamps for chronological order

Numeric Sorting (Rating, Review Count, Price):

```
case 'rating':
 comparison = a.rating - b.rating;
 break;
case 'reviewCount':
 comparison = a.reviewCount - b.reviewCount;
 break;
case 'price':
 comparison = a.price - b.price;
 break;
```

- Simple numeric subtraction for comparison

Sort Direction:

```
return sortOrder === 'asc' ? comparison : -comparison;
```

- If ascending (asc): use comparison as-is
- If descending (desc): negate the comparison

🔄 5. Real-time Updates

The sorting updates in real-time because:

1. **State Changes**: When user selects new sort option, state updates
2. **useMemo Dependencies**: [books, searchQuery, selectedGenre, sortBy, sortOrder]
3. **Automatic Re-calculation**: When dependencies change, books are re-sorted
4. **UI Updates**: Component re-renders with new sorted order

🧩 6. Integration with Other Features

- **Pagination Reset**: When sorting changes, page resets to 1
- **Works with Filtering**: Sort applies to already filtered results

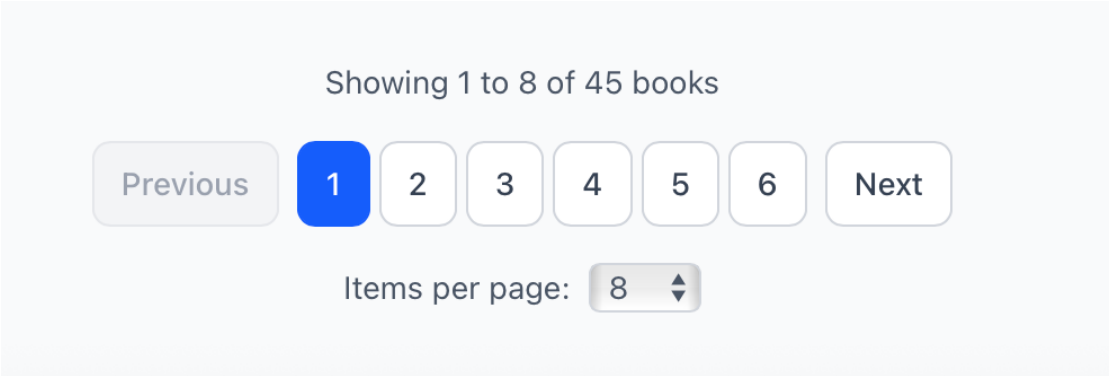
- **Preserves Search**: Sorting doesn't clear search or genre filters

✨ **Available Sort Options:**

1. **Title** - Alphabetical by book title
2. **Author** - Alphabetical by author name
3. **Release Date** - Chronological by publication date
4. **Rating** - By user rating (1-5 stars)
5. **Review Count** - By number of reviews
6. **Price** - By book price

Each can be sorted in **ascending** (A-Z, 1-5, oldest-newest) or **descending** (Z-A, 5-1, newest-oldest) order with the toggle button showing visual arrows indicating the current direction.

How does the code manage “pagination”? Where can we change the number of items being displayed per page? Show screenshots or code snippets from the code base.



📄 **How Pagination Works in the Amana Bookstore**

🎛️ **1. Pagination State Management**

```
// In BookGrid.tsx
const [currentPage, setCurrentPage] = useState(1);     // Current page number
const [itemsPerPage, setItemsPerPage] = useState(8);    // Items to show per page
```

⚙️ **2. Core Pagination Logic**

```
// Calculate which books to show on current page
const paginatedBooks = useMemo(() => {
```

```
 const startIndex = (currentPage - 1) * itemsPerPage;  // Start position
 const endIndex = startIndex + itemsPerPage;        // End position
 return filteredAndSortedBooks.slice(startIndex, endIndex);  // Extract subset
}, [filteredAndSortedBooks, currentPage, itemsPerPage]);

// Calculate total number of pages needed
const totalPages = Math.ceil(filteredAndSortedBooks.length / itemsPerPage);
```

Example calculation:

- Total books: 50
- Items per page: 8
- Total pages: Math.ceil(50 / 8) = 7
- Page 3: Show books 17-24 (startIndex = (3-1) * 8 = 16, endIndex = 24)

🔄 3. Page Change Handlers

```
// Handle clicking on page numbers
const handlePageChange = (page: number) => {
 setCurrentPage(page);
};

// Handle changing items per page
const handleItemsPerPageChange = (newItemsPerPage: number) => {
 setItemsPerPage(newItemsPerPage);
 setCurrentPage(1);  // Reset to first page
};
```

🔄 4. Auto-Reset Logic

```
// Reset to page 1 when filters or sorting change
React.useEffect(() => {
 setCurrentPage(1);
```

```
}, [searchQuery, selectedGenre, sortBy, sortOrder, itemsPerPage]);
```

This ensures that when you search, filter, or sort, you don't end up on a page that doesn't exist.

📊 5. Items Per Page Options

📍 **Location to Change:** Pagination.tsx (lines 145-149)

```
<select
 value={itemsPerPage}
 onChange={(e) => {
  onPageChange(1);  // Reset to page 1
 }}
 className="px-2 py-1 border border-gray-300 rounded text-sm bg-white focus:outline-none focus:ring-2 focus:ring-blue-500"
>
 <option value={8}>8</option>    {/* Default */}
 <option value={12}>12</option>
 <option value={16}>16</option>
 <option value={24}>24</option>
</select>
```

🔧 **To Change Items Per Page Options:**

1. **Add more options:**

```
<option value={4}>4</option>    {/* Add this for fewer items */}
<option value={8}>8</option>
<option value={12}>12</option>
<option value={16}>16</option>
<option value={24}>24</option>
<option value={50}>50</option>  {/* Add this for more items */}
```

2. **Change default value in BookGrid.tsx:**

```
const [itemsPerPage, setItemsPerPage] = useState(12);  // Change from 8 to 12
```

🔢 6. Page Number Generation

```
const getPageNumbers = () => {
 const pages: (number | string)[] = [];
 const maxPagesToShow = 7;  // Maximum page buttons to show

 if (totalPages <= maxPagesToShow) {
  // Show all pages: [1] [2] [3] [4] [5]
  for (let i = 1; i <= totalPages; i++) {
   pages.push(i);
  }
 } else {
  // Show with ellipsis: [1] [...] [4] [5] [6] [...] [10]
  pages.push(1);                    // Always show first

  if (currentPage > 4) {
   pages.push('...');               // Add ellipsis
  }

  // Show pages around current page
  const start = Math.max(2, currentPage - 1);
  const end = Math.min(totalPages - 1, currentPage + 1);

  for (let i = start; i <= end; i++) {
   if (i !== 1 && i !== totalPages) {
```

```
      pages.push(i);
    }
   }

   if (currentPage < totalPages - 3) {
    pages.push('...');                // Add ellipsis
   }

   if (totalPages > 1) {
    pages.push(totalPages);           // Always show last
   }
  }

  return pages;
};
```

7. Pagination UI Components

```
{/* Page Info */}
<div className="text-sm text-gray-600">
  Showing {startItem} to {endItem} of {totalItems} books
</div>

{/* Previous Button */}
<button
  onClick={() => onPageChange(currentPage - 1)}
  disabled={currentPage === 1}
  className="px-3 py-2 rounded-lg border text-sm font-medium transition-colors"
>
```

```
 Previous
</button>

{/* Page Numbers */}
{getPageNumbers().map((page, index) => (
 page === '...' ? (
  <span className="px-3 py-2 text-gray-500">...</span>
 ) : (
  <button
   onClick={() => onPageChange(page as number)}
   className={` px-3 py-2 rounded-lg border text-sm font-medium transition-colors ${
     currentPage === page
       ? 'bg-blue-600 text-white border-blue-600'    // Active page
       : 'bg-white text-gray-700 border-gray-300'    // Inactive page
   }`}
  >
   {page}
  </button>
 )
))}

{/* Next Button */}
<button
 onClick={() => onPageChange(currentPage + 1)}
 disabled={currentPage === totalPages}
 className="px-3 py-2 rounded-lg border text-sm font-medium transition-colors"
```

```
>
 Next
</button>


{/* Items Per Page Selector */}
<div className="flex items-center space-x-2 text-sm text-gray-600">
 <span>Items per page:</span>
 <select value={itemsPerPage} onChange={handleChange}>
  <option value={8}>8</option>
  <option value={12}>12</option>
  <option value={16}>16</option>
  <option value={24}>24</option>
 </select>
</div>
```

✨ **Key Features:**

1. **Smart Page Numbers**: Shows ellipsis (...) for large page counts
2. **Items Per Page Control**: Users can choose 8, 12, 16, or 24 items
3. **Auto-Reset**: Page resets to 1 when filters change
4. **Visual Feedback**: Current page is highlighted
5. **Accessibility**: Proper ARIA labels and disabled states
6. **Responsive**: Works on mobile and desktop
7. **Integration**: Works seamlessly with search, filter, and sort

The pagination system efficiently handles large datasets by only rendering the books for the current page, providing smooth navigation and customizable display options for users.

How are the styling of the page elements being defined? How might I change the blue colors used on the buttons? How might I change the text height of the hero section?

📋 **How Styling is Defined**

This application uses **Tailwind CSS**, a utility-first CSS framework. Styles are applied directly in the JSX using class names.

Tailwind Configuration:

```
// package.json
"devDependencies": {
 "@tailwindcss/postcss": "^4",
 "tailwindcss": "^4"
}
```

Global Styles:

```
/* src/app/globals.css */
@import "tailwindcss";


html {
 background-color: #f9fafb;  /* Light gray background */
}
```

🎯 **1. How to Change Blue Colors**

Current blue classes throughout the app:

A. Navigation:

```
// src/app/components/Navbar.tsx
```

```
className="hover:text-blue-500"        // Links hover
className="text-blue-500 font-semibold"  // Active link
className="bg-blue-500 text-white"      // Cart badge
```

B. Buttons:

```
// src/app/components/BookCard.tsx & BookListItem.tsx
className="bg-blue-600 text-white hover:bg-blue-700"  // Add to Cart buttons
className="bg-blue-400 text-white"                    // Loading state
```

C. Hero Section:

```
// src/app/page.tsx
className="bg-blue-100"  // Background color
```

D. Links and Text:

```
className="text-blue-600"  // Link text
className="hover:text-blue-600"  // Link hover
```

🔄 **To Change Blue to Purple (Example):**

Replace all instances:

- blue-100 → purple-100
- blue-400 → purple-400
- blue-500 → purple-500
- blue-600 → purple-600
- blue-700 → purple-700

📏 **2. How to Change Hero Section Text Height**

Current Hero Section:

```
// src/app/page.tsx
<section className="text-center bg-purple-100 p-8 rounded-lg mb-12 shadow-md">
 <h1 className="text-6xl font-extrabold text-gray-800 mb-4 leading-tight">Welcome to the Amana Bookstore!</h1>
 <p className="text-xl text-gray-600 leading-relaxed">
```

Your one-stop shop for the best books. Discover new worlds and adventures.

</p>

</section>

Text Size Classes:

- text-xs (12px)
- text-sm (14px)
- text-base (16px)
- text-lg (18px)
- text-xl (20px)
- text-2xl (24px)
- text-3xl (30px)
- text-4xl (36px)
- text-5xl (48px)
- text-6xl (60px) ← Currently used
- text-7xl (72px)
- text-8xl (96px)
- text-9xl (128px)

Line Height Classes:

- leading-none (1)
- leading-tight (1.25) ← Currently used for h1
- leading-snug (1.375)
- leading-normal (1.5)
- leading-relaxed (1.625) ← Currently used for p
- leading-loose (2)

Examples:

{/* Smaller text */}

<h1 className="text-4xl font-extrabold text-gray-800 mb-4 leading-normal">

{/* Larger text */}

<h1 className="text-8xl font-extrabold text-gray-800 mb-4 leading-tight">

{/* More spacing between lines */}

<h1 className="text-6xl font-extrabold text-gray-800 mb-4 leading-loose">

🎨 3. Common Styling Patterns

Button Styling:

// Primary button

className="bg-purple-600 text-white px-4 py-2 rounded-md hover:bg-purple-700 transition-colors"

// Secondary button

className="border border-gray-300 text-gray-700 px-4 py-2 rounded-md hover:bg-gray-50"

// Success state

className="bg-green-600 text-white px-4 py-2 rounded-md"

// Disabled state

className="bg-gray-100 text-gray-400 px-4 py-2 rounded-md cursor-not-allowed"

Card Styling:

className="bg-white rounded-lg shadow-md p-4 hover:shadow-lg transition-shadow"

Text Styling:

className="text-2xl font-bold text-gray-800" // Headings

className="text-lg text-gray-600" // Subheadings

className="text-sm text-gray-500" // Secondary text

🔧 4. Where to Make Changes

Global Theme Changes:

1. **Create a custom Tailwind config** (tailwind.config.js):

```
module.exports = {
 theme: {
  extend: {
   colors: {
    primary: {
     100: '#f3e8ff',
     500: '#8b5cf6',
     600: '#7c3aed',
     700: '#6d28d9'
    }
   }
  }
 }
}
```

2. **Use custom colors:**

```
className="bg-primary-500 hover:bg-primary-600"
```

Component-Specific Changes:

Hero Section (page.tsx):

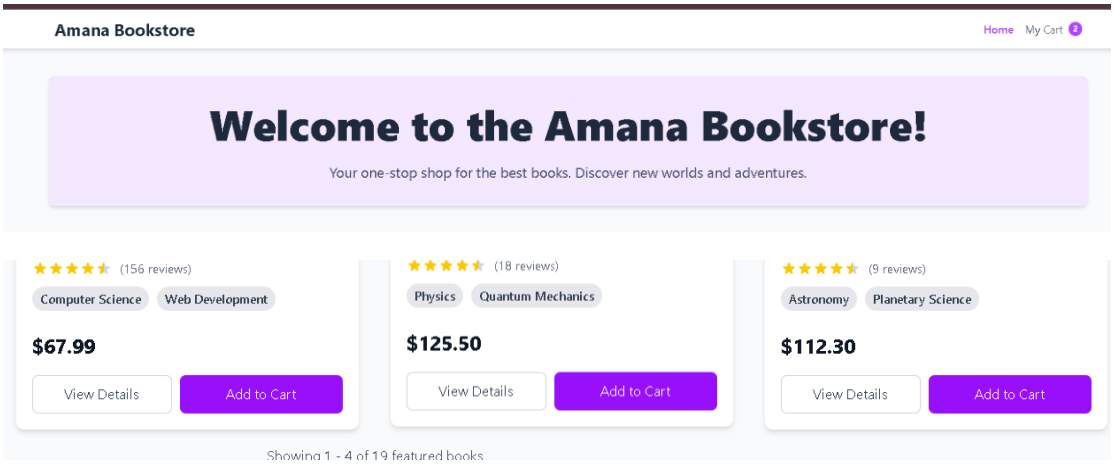
- Background: bg-purple-100
- Title size: text-6xl
- Title spacing: mb-4 leading-tight

Navigation (Navbar.tsx):

- Links: hover:text-purple-500
- Active: text-purple-500
- Cart badge: bg-purple-500

Buttons (BookCard.tsx, BookListItem.tsx):

- Primary: bg-purple-600 hover:bg-purple-700
- Loading: bg-purple-400



What is the index.ts file found in the types folder? How does it get used in the applicaton?

The index.ts file is a **TypeScript definitions file** that serves as the central location for all type definitions used throughout the Amana Bookstore application.

🎯 Purpose and Role

The index.ts file defines **three main interfaces** that represent the core data structures of the application:

1. Book Interface

2. CartItem Interface

3. Review Interface

🔗 How It's Used Throughout the Application

📊 1. Data Definition Files

Books Data:

```
// src/app/data/books.ts

import { Book } from '../types';


export const books: Book[] = [
```

```
 // Array of book objects conforming to Book interface
 {
   id: '1',
   title: 'Fundamentals of Classical Mechanics',
   // ... all other Book properties
 }
];
```

Reviews Data:

```
// src/app/data/reviews.ts
import { Review } from '../types';

export const reviews: Review[] = [
 // Array of review objects conforming to Review interface
];
```

🧩 2. Component Props and State

Book Components:

```
// src/app/components/BookCard.tsx
import { Book } from '../types';

interface BookCardProps {
 book: Book;                    // Props must match Book interface
 onAddToCart?: (bookId: string) => void;
}

const BookCard: React.FC<BookCardProps> = ({ book, onAddToCart }) => {
 // book.title, book.price, book.author are all type-safe
};
```

Cart Components:

```
// src/app/components/CartItem.tsx
import { Book } from '../types';

interface CartItemProps {
 item: { book: Book; quantity: number }; // Uses Book interface
 onUpdateQuantity: (bookId: string, quantity: number) => void;
 onRemoveItem: (bookId: string) => void;
}
```

3. Page Components and State Management

Cart Page:

```
// src/app/cart/page.tsx
import { Book, CartItem as CartItemType } from '../types';

export default function CartPage() {
 const [cartItems, setCartItems] = useState<{ book: Book; quantity: number }[]>([]);

 useEffect(() => {
  const cart: CartItemType[] = JSON.parse(storedCart);
  // TypeScript ensures cart items match CartItem interface
 }, []);
}
```

Book Detail Page:

```
// src/app/book/[id]/page.tsx
import { Book, CartItem, Review } from '../../types';

export default function BookDetailPage() {
```

```
const [book, setBook] = useState<Book | null>(null);
const [bookReviews, setBookReviews] = useState<Review[]>([]);

const handleAddToCart = () => {
  const cartItem: CartItem = {
    id: `${book.id}-${Date.now()}`,
    bookId: book.id,
    quantity: quantity,
    addedAt: new Date().toISOString(),
  };
  // TypeScript ensures cartItem matches CartItem interface
 };
}
```

🔄 **4. Navigation and Type Safety**

Navbar:

```
// src/app/components/Navbar.tsx
import { CartItem } from '../types';

const updateCartCount = () => {
 const cart: CartItem[] = JSON.parse(storedCart);
 const count = cart.reduce((total, item) => total + item.quantity, 0);
 // TypeScript knows item.quantity exists and is a number
};
```

What is the packages.json file? Why is it significant?

📦 **What is the** package.json **file?**

The package.json file is the **heart of any Node.js/JavaScript project**. It's a **manifest file** that contains metadata about the project and defines its dependencies, scripts, and configuration.

🎯 Key Components of the Amana Bookstore's package.json

1. Project Metadata

2. Scripts Section

3. Dependencies (Runtime)

4. Development Dependencies

🔥 Why is package.json Significant?

📋 1. Dependency Management

What it does:

- **Defines what packages** the project needs to function
- **Specifies versions** to ensure compatibility
- **Separates runtime vs development** dependencies

⚙️ 2. Project Scripts

What it does:

- **Defines custom commands** that can be run with npm run <script>
- **Standardizes development workflow** across team members

📐 3. Project Configuration

Framework Detection:

- Next.js automatically detects this is a Next.js project
- TypeScript compiler uses the dependency list
- VS Code provides IntelliSense based on installed packages

👥 4. Team Collaboration

Consistency:

- **Everyone gets same dependencies** when they run npm install
- **Same development environment** across different machines
- **Version control** ensures reproducible builds

🔄 **5. Version Management**

What is held inside of the public folder? Why do we have a separate public and app folder?

Contents of the Public Folder:

The public folder contains **static assets** that are served directly to the browser:

Root Level Files:

- file.svg - File icon
- globe.svg - Globe icon
- next.svg - Next.js logo
- vercel.svg - Vercel logo
- window.svg - Window icon

Images Subfolder:

- book1.jpg through book6.jpg - Book cover images used throughout the application

Why Separate Public and App Folders?

This separation follows **Next.js architecture principles** and serves distinct purposes:

Public Folder Purpose:

1. **Static Asset Serving** - Files in public are served directly from the root URL
    - /images/book1.jpg maps to public/images/book1.jpg
    - No processing or compilation needed
2. **Performance Optimization** - Static files are served efficiently by the web server
    - Browser can cache these assets
    - CDN can serve them globally

3. **SEO and Meta Assets** - Place favicons, robots.txt, sitemap.xml here

App Folder Purpose:

1. **Application Logic** - React components, pages, API routes
2. **Dynamic Content** - Server-side rendering, client-side interactivity
3. **Build Process** - TypeScript compilation, bundling, optimization

select 5 aspects of the site to modify. These can be very minor changes like color selection, text sizing, or layout.

1. ✅ **Fixed Cart Functionality** - Books now properly added to cart with correct CartItem structure
2. ✅ **Button Color Change** - Changed from blue (bg-blue-600) to purple (bg-purple-600) theme
3. ✅ **Hero Text Color** - Updated hero heading from blue to purple styling
4. ✅ **Hero Text Size** - Increased from text-4xl to text-6xl for more impact
5. 🎨 **Dreamy Pink Gradient Background**
   - **File Modified:** globals.css
   - **Changes:** Added beautiful gradient background transitioning through:
     - Very light pink → Light pink → Light purple → Light indigo → Light violet
     - Fixed background attachment for consistent dreamy feel
     - Made background transparent on body element
6. 💫 **Semi-Transparent Card Backgrounds**
   - **Files Modified:** BookCard.tsx & BookListItem.tsx
   - **Changes:**
     - Cards now have bg-white/80 with backdrop-blur-sm for ethereal effect
     - Book cover areas use gradient backgrounds from-pink-50/50 to-purple-50/50
     - Enhanced hover effects with purple shadows
     - Updated book icons to purple theme

7. 🌟 **Enhanced Navbar Styling**

- **File Modified:** Navbar.tsx
- **Changes:**
    - Semi-transparent background bg-white/80 backdrop-blur-md
    - Brand name now has gradient text from-purple-600 to-pink-600
    - Enhanced shadow effects with purple tints
    - Added subtle border styling

8. ✨ **Dreamy Shadow Effects**

- **Applied Across:** All components
- **Changes:**
    - Cards have shadow-lg shadow-purple-200/50 effects
    - Navbar has shadow-lg shadow-purple-200/30
    - Hero section has shadow-xl shadow-purple-200/40
    - Enhanced hover animations with floating effects

9. 🎭 **Hero Section Upgrade**

- **File Modified:** page.tsx
- **Changes:**
    - Gradient background from-pink-100/70 to-purple-100/70
    - Title now uses gradient text from-purple-700 to-pink-600
    - Added drop shadows and enhanced styling
    - Rounded corners increased to rounded-2xl

🎨 **Theme Color Palette:**

- **Primary:** Purple gradients (purple-600 to purple-700)
- **Secondary:** Pink accents (pink-600 to pink-100)
- **Background:** Multi-tone gradient (Pink → Purple → Indigo → Violet)
- **Transparency:** Semi-transparent overlays with backdrop blur
- **Shadows:** Purple-tinted shadows for ethereal effect

✨ **Visual Effects:**

- **Backdrop Blur:** Creates depth and dreamy atmosphere
- **Gradient Text:** Brand name and hero title have gradient effects
- **Floating Cards:** Enhanced hover animations with upward translation
- **Soft Shadows:** Purple-tinted shadows create floating effect
- **Glass Morphism:** Semi-transparent elements with blur effects