i have created e-library website. in here have [bookId]/page. In here, after user click heart icon that books should add to favorite book to that user. So that book details should goes to user profile's favorite section.

This is categories/[category]/[bookId]/page. Update heart icon to do task

```
"use client";
import { useState, useEffect } from 'react';
import { useSession } from 'next-auth/react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faHeart, faDownload } from '@fortawesome/free-solid-svg-icons';
import axios from 'axios';

export default function BookPage({ params }) {
  const { category, bookId } = params;
  const [book, setBook] = useState(null);
  const [isFavorite, setIsFavorite] = useState(false);  // Track if the
book is favorited
  const [loading, setLoading] = useState(true); // Track loading state
  const { data: session } = useSession();

  useEffect(() => {
    const fetchBookDetails = async () => {
      try {
        const { data } = await axios.get(`/api/books/${bookId}`);
        setBook(data.book);
        setLoading(false);
      } catch (error) {
        console.error('Failed to fetch book details:', error.message);
      }
    };

    const checkIfFavorite = async () => {
      if (session) {
        try {
          const { data } = await
axios.get(`/api/user/favorite?email=${session.user.email}`);
          const favoriteBooks = data.favorites.map(fav => fav._id); //
Extract book IDs
          setIsFavorite(favoriteBooks.includes(bookId)); // Check if
current book is in favorites
```

```
        } catch (error) {
          console.error('Failed to check favorites:', error.message);
        }
      }
    };

    fetchBookDetails();
    checkIfFavorite();
  }, [bookId, session]);

  const handleDownload = async () => {
    try {
      if (session) {
        await axios.post('/api/user/download', {
          email: session.user.email,
          bookId: bookId,
          bookTitle: book.title,
        });

        // Trigger the download
        const link = document.createElement('a');
        link.href = book.pdfUrl;
        link.setAttribute('download', book.title);
        document.body.appendChild(link);
        link.click();
        link.remove();
      } else {
        alert('You need to be logged in to download the book.');
      }
    } catch (error) {
      console.error('Failed to update user downloads:', error.message);
    }
  };
  const handleAddToFavorites = async () => {
    try {
      if (session) {
        const response = await axios.post('/api/user/favorite', {
          email: session.user.email, // User's email
          bookId: bookId, // Ensure bookId is passed correctly here
        });
        if (response.status === 200) {
          setIsFavorite(true); // Update favorite state
          alert('Book added to favorites successfully!');
        }
      } else {
        alert('You need to be logged in to add a book to favorites.');
      }
    } catch (error) {
      console.error('Failed to add book to favorites:', error);
      alert('Failed to add book to favorites.');
    }
  };


  if (loading) {
    return <p>Loading...</p>;
  }

  if (!book) {
    return <p>Failed to load book details.</p>;
  }

  return (
    <div className="relative min-h-screen p-6 bg-gray-50 overflow-hidden">
      {/* Background image styling */}
      <div
        style={{
          position: 'absolute',
          inset: 0,
          backgroundImage: `url(${book.backgroundImageUrl ||
'/back.jpg'})`,
          backgroundSize: 'cover',
          backgroundPosition: 'center',
          backdropFilter: 'blur(10px)',
        }}
      >
        <div
          style={{
            position: 'absolute',
            inset: 0,
            backgroundColor: 'rgba(0, 0, 0, 0.5)',
          }}
        />
      </div>

      <div className="relative z-10 px-8 py-12 bg-gray-50 min-h-screen">
        {/* Back to category navigation */}
        <div className="bg-gray-100 py-2 px-4 mb-8 rounded-lg shadow-md">
          <nav className="text-gray-600 text-sm font-semibold">
            <a href={`/categories/${category}`}
className="hover:text-gray-800">
              BACK TO {category.toUpperCase()}
            </a>
          </nav>
        </div>

        {/* Book details section */}
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-12 bg-white
p-8 rounded-xl shadow-lg">
          {/* Book cover image */}
          <div className="flex justify-center">
            <img
              src={book.coverImageUrl}
              alt={book.title}
              className="w-2/3 lg:w-1/3 h-auto object-contain rounded-lg"
            />
          </div>
          {/* Book details */}
          <div className="flex flex-col justify-between">
            <div>
              <h1 className="text-4xl font-semibold text-gray-900
mb-6">{book.title}</h1>
              <p className="text-lg text-gray-700
mb-6">{book.description}</p>
              <p className="text-sm text-gray-600 mb-4">
                Author: <span className="font-medium
text-gray-900">{book.author}</span>
              </p>
              <p className="text-sm text-gray-600">
```

```
                Category: <span className="font-medium
text-gray-900">{category}</span>
              </p>
            </div>

            {/* Buttons for adding to favorites and downloading */}
            {session && (
              <div className="flex space-x-6 mt-6">
                {/* Heart icon button for adding to favorites */}
                <button
  onClick={handleAddToFavorites}
  className={`flex items-center justify-center w-12 h-12
  rounded-full shadow-md transition duration-200 ${
    isFavorite ? 'bg-green-500' : 'bg-red-500 hover:bg-red-600'
  }`}
>
  <FontAwesomeIcon icon={faHeart} className="w-6 h-6 text-white" />
</button>


                {/* Download button */}
                <button
                  onClick={handleDownload}
                  className="flex items-center justify-center w-12 h-12
bg-blue-500 text-white rounded-full shadow-md hover:bg-blue-600 transition
duration-200"
                >
                  <FontAwesomeIcon icon={faDownload} className="w-6 h-6"
/>
                </button>
              </div>
            )}
          </div>
        </div>
      </div>
    </div>
  );
}
```

This is api/bookId/route

```
import { NextResponse } from 'next/server';
import multer from 'multer';
import { promisify } from 'util';
import fs from 'fs';
import path from 'path';
import connect from '../../../../utils/db';
import Book from '../../../../models/Book';

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadPath = 'public/uploads';
    if (!fs.existsSync(uploadPath)) {
      fs.mkdirSync(uploadPath, { recursive: true });
    }
    cb(null, uploadPath);
  },
  filename: (req, file, cb) => {
    const sanitizedFileName = file.originalname.replace(/[^a-zA-Z0-9.-]/g,
'_').replace(/\s+/g, '_');
    cb(null, `${Date.now()}-${sanitizedFileName}`);
  },
});

const upload = multer({ storage: storage });
const uploadMiddleware = promisify(upload.fields([{ name: 'coverImage' },
{ name: 'pdf' }]));

// GET: Fetch a single book by ID
export const GET = async (req, { params }) => {
  await connect();
  const { id } = params;  // Extract bookId from the route params

  if (!id) {
    return NextResponse.json({ error: 'Book ID is required' }, { status:
400 });
```

```
  }

  try {
    const book = await Book.findById(id).exec();

    if (!book) {
      return NextResponse.json({ error: 'Book not found' }, { status: 404
});
    }

    return NextResponse.json({ book }, { status: 200 });
  } catch (error) {
    console.error('Error fetching book:', error);
    return NextResponse.json({ error: 'Failed to fetch book' }, { status:
500 });
  }
};

// PUT: Update book by ID
export const PUT = async (req, { params }) => {
  try {
    // Handle file uploads
    await uploadMiddleware(req, null);

    const bookId = params.id; // Extract bookId from the URL path

    if (!bookId) {
      console.error('Book ID missing in PUT request');
      return NextResponse.json({ error: 'Book ID is required' }, { status:
400 });
    }

    // Extract form data
    const formData = await req.formData();
    const { title, author, description, category } =
Object.fromEntries(formData.entries());

    if (!title || !author || !description || !category) {
      console.error('Required book fields are missing');
      return NextResponse.json({ error: 'All book fields are required' },
{ status: 400 });
    }

    const updatedData = {
      title,
      author,
      description,
      category,
    };

    // Handle uploaded files if present
    if (formData.has('coverImage')) {
      const coverImage = formData.get('coverImage');
      updatedData.coverImageUrl = `/uploads/${coverImage.name}`;
    }
    if (formData.has('pdf')) {
      const pdf = formData.get('pdf');
      updatedData.pdfUrl = `/uploads/${pdf.name}`;
    }

    // Update the book in the database
    const updatedBook = await Book.findByIdAndUpdate(bookId, updatedData,
{ new: true });

    if (!updatedBook) {
      console.error('Book not found in database');
      return NextResponse.json({ error: 'Book not found' }, { status: 404
});
    }

    return NextResponse.json({ message: 'Book updated successfully', book:
updatedBook }, { status: 200 });
  } catch (error) {
    console.error('Error updating book:', error);
    return NextResponse.json({ error: 'Failed to update book' }, { status:
500 });
  }
};
```

```
// DELETE: Delete book by ID
export const DELETE = async (req, { params }) => {
  await connect();

  try {
    const bookId = params.id; // Capture book ID from URL path

    if (!bookId) {
      console.error('Book ID is missing');
      return NextResponse.json({ error: 'Book ID is required' }, { status:
400 });
    }

    // Log the bookId to ensure it is correct
    console.log(`Deleting book with ID: ${bookId}`);

    // Find the book in the database
    const book = await Book.findById(bookId);

    if (!book) {
      console.error(`Book with ID ${bookId} not found`);
      return NextResponse.json({ error: 'Book not found' }, { status: 404
});
    }

    // Optionally delete associated files (coverImage and PDF)
    if (book.coverImageUrl) {
      const coverImagePath = path.join(process.cwd(), 'public',
book.coverImageUrl);
      if (fs.existsSync(coverImagePath)) {
        fs.unlinkSync(coverImagePath);
      }
    }

    if (book.pdfUrl) {
      const pdfPath = path.join(process.cwd(), 'public', book.pdfUrl);
      if (fs.existsSync(pdfPath)) {
        fs.unlinkSync(pdfPath);
      }
    }
    // Delete the book from the database
    await Book.findByIdAndDelete(bookId);

    console.log(`Book with ID ${bookId} deleted successfully`);
    return NextResponse.json({ message: 'Book deleted successfully' }, {
status: 200 });
  } catch (error) {
    console.error('Error deleting book:', error);
    return NextResponse.json({ error: 'Failed to delete book' }, { status:
500 });
  }
};
```

This is api/favorite/[email]/route

```
import { NextResponse } from 'next/server';
import dbConnect from '../../../../utils/db';
import User from '../../../../models/User';
import Book from '../../../../models/Book';

// GET: Fetch all favorite books for a user by email
export async function GET(req) {
  try {
    const { searchParams } = new URL(req.url);
    const email = searchParams.get('email');

    if (!email) {
      return NextResponse.json({ message: 'Email is required' }, { status:
400 });
    }

    await dbConnect();

    const user = await User.findOne({ email
}).populate('favorites.bookId');
    if (!user) {
      return NextResponse.json({ message: 'User not found' }, { status:
404 });
```

```
    }

    return NextResponse.json({ favorites: user.favorites }, { status: 200
});
  } catch (error) {
    console.error('Error fetching favorites:', error);
    return NextResponse.json({ message: 'Server error' }, { status: 500
});
  }
}

// POST: Add a book to the user's favorites
export async function POST(req) {
  await dbConnect();

  try {
    const { email, bookId } = await req.json(); // Extract email and
bookId from the request body

    if (!email || !bookId) {
      return NextResponse.json({ error: 'Email and Book ID are required'
}, { status: 400 });
    }

    // Find the user by email
    const user = await User.findOne({ email });

    if (!user) {
      return NextResponse.json({ error: 'User not found' }, { status: 404
});
    }

    // Check if the book exists
    const book = await Book.findById(bookId);

    if (!book) {
      return NextResponse.json({ error: 'Book not found' }, { status: 404
});
    }
    // Initialize favorites if undefined
    if (!user.favorites) {
      user.favorites = [];
    }

    // Check if the book is already in the user's favorites
    const alreadyFavorite = user.favorites.some(favorite =>
favorite.equals(bookId));

    if (alreadyFavorite) {
      return NextResponse.json({ message: 'Book is already in favorites'
}, { status: 200 });
    }

    // Add the book to the user's favorites
    user.favorites.push(bookId);
    await user.save();

    return NextResponse.json({ message: 'Book added to favorites
successfully' }, { status: 200 });
  } catch (error) {
    console.error('Failed to add book to favorites:', error);
    return NextResponse.json({ error: 'Failed to add book to favorites' },
{ status: 500 });
  }
}
```

This is api/favorite/route

```
import { NextResponse } from 'next/server';
import dbConnect from '../../../../utils/db';
import User from '../../../../models/User';
import Book from '../../../../models/Book';

// GET: Fetch all favorite books for a user by email
```

```
export async function GET(req) {
  try {
    const { searchParams } = new URL(req.url);
    const email = searchParams.get('email');

    if (!email) {
      return NextResponse.json({ message: 'Email is required' }, { status:
400 });
    }

    await dbConnect();

    const user = await User.findOne({ email
}).populate('favorites.bookId');
    if (!user) {
      return NextResponse.json({ message: 'User not found' }, { status:
404 });
    }

    return NextResponse.json({ favorites: user.favorites }, { status: 200
});
  } catch (error) {
    console.error('Error fetching favorites:', error);
    return NextResponse.json({ message: 'Server error' }, { status: 500
});
  }
}

// POST: Add a book to the user's favorites
export async function POST(req) {
  await dbConnect();

  try {
    const { email, bookId } = await req.json(); // Extract email and
bookId

    if (!email || !bookId) {
      return NextResponse.json({ error: 'Email and Book ID are required'
}, { status: 400 });
    }
    // Check if the book exists
    const book = await Book.findById(bookId);
    if (!book) {
      return NextResponse.json({ error: 'Book not found' }, { status: 404
});
    }

    // Find the user by email
    const user = await User.findOne({ email });
    if (!user) {
      return NextResponse.json({ error: 'User not found' }, { status: 404
});
    }

    // Check if the book is already in favorites
    const alreadyFavorite = user.favorites.some(favorite =>
favorite.equals(bookId));
    if (alreadyFavorite) {
      return NextResponse.json({ message: 'Book is already in favorites'
}, { status: 200 });
    }

    // Add the book to the user's favorites
    user.favorites.push(bookId);
    await user.save();

    return NextResponse.json({ message: 'Book added to favorites
successfully' }, { status: 200 });
  } catch (error) {
    console.error('Failed to add book to favorites:', error);
    return NextResponse.json({ error: 'Failed to add book to favorites' },
{ status: 500 });
  }
}
```

This is user model

```
import mongoose from "mongoose";

const { Schema } = mongoose;

// Define a schema for the Favorite Books

const userSchema = new Schema(
  {
    email: {
      type: String,
      unique: true,
      required: true,
    },
    password: {
      type: String,
      required: false,
    },
    role: {
      type: String,
      enum: ['user', 'admin'],
      default: 'user',
    },
    name: {
      type: String,
      required: false,
    },
    country: {
      type: String,
      required: false,
    },
    favoriteBook: {
      type: String,
      required: false,
    },
    profilePhoto: {
      type: String, // URL to the photo
      required: false,
    },
    downloads: [
      {
        bookId: {
          type: mongoose.Schema.Types.ObjectId,
          ref: 'Book',
          required: true
        }, // Reference to Book model
        downloadedAt: {
          type: Date,
          default: Date.now
        }, // Store download timestamp
      },
    ],
    favorites: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Book' }], // Array of book IDs

  },
  { timestamps: true }
);

export default mongoose.models.User || mongoose.model("User", userSchema);
```

This is book model

```
import mongoose from 'mongoose';

const BookSchema = new mongoose.Schema({
  title: { type: String, required: true },
  author: { type: String, required: true },
  description: { type: String, required: true },
  category: { type: String, required: true },
  coverImageUrl: { type: String },
  pdfUrl: { type: String },
  adminEmail: { type: String, required: true }, // To relate the book to the admin
  createdAt: { type: Date, default: Date.now },
});

export default mongoose.models.Book || mongoose.model('Book', BookSchema);
```

This is user profile. Which have favorite section

```
"use client";

import { useSession } from 'next-auth/react';
import { useState, useEffect } from 'react';
import axios from 'axios';

export default function UserProfile() {
  const { data: session } = useSession();
  const [selectedTab, setSelectedTab] = useState('profile');
  const [isEditing, setIsEditing] = useState(false);
  const [profileData, setProfileData] = useState({
    name: '',
    email: '',
    country: '',
    favoriteBook: '',
    profilePhoto: '',
  });
  const [downloads, setDownloads] = useState([]);
  const [favorites, setFavorites] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  // Fetch user profile data
  useEffect(() => {
    if (session) {
      const fetchUserProfile = async () => {
        try {
          const response = await axios.get('/api/updateProfile', {
            params: { email: session.user.email },
          });
          setProfileData(response.data.user);
        } catch (error) {
          console.error('Error fetching profile:', error);
          setError('Failed to load profile. Please try again.');
        }
      };
      fetchUserProfile();
    }
  }, [session]);

  // Fetch user's downloads
  useEffect(() => {
    if (selectedTab === 'downloads' && session) {
      const fetchDownloads = async () => {
        try {
          const response = await axios.get('/api/user/downloads', {
            params: { email: session.user.email },
          });
          setDownloads(response.data.downloads);
        } catch (error) {
          console.error('Error fetching downloads:', error);
          setError('Failed to load downloads. Please try again.');
        }
      };
      fetchDownloads();
    }
  }, [selectedTab, session]);

  // Fetch user's favorite books
  useEffect(() => {
    if (selectedTab === 'favorites' && session) {
      const fetchFavorites = async () => {
        try {
          const response = await
axios.get(`/api/favorite?email=${session.user.email}`);
          setFavorites(response.data.favorites);
        } catch (error) {
          console.error('Error fetching favorites:', error);
          setError('Failed to load favorites. Please try again.');
        }
      };

      fetchFavorites();
    }
```

```
}, [selectedTab, session]);

if (!session) {
  return <p>Loading...</p>;
}

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setProfileData((prevData) => ({ ...prevData, [name]: value }));
};

const handlePhotoChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    setProfileData((prevData) => ({ ...prevData, profilePhoto: file }));
  }
};

const handleSave = async () => {
  setLoading(true);
  setError('');

  const formData = new FormData();
  formData.append('name', profileData.name);
  formData.append('email', profileData.email);
  formData.append('country', profileData.country);
  formData.append('favoriteBook', profileData.favoriteBook);

  if (profileData.profilePhoto instanceof File) {
    formData.append('profilePhoto', profileData.profilePhoto);
  }

  try {
    const response = await axios.put('/api/updateProfile', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });

    setProfileData((prevData) => ({
        ...prevData,
        profilePhoto: response.data.user.profilePhoto,
      }));
      setIsEditing(false);
    } catch (error) {
      console.error('Error updating profile:', error);
      setError('Failed to update profile. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  const renderContent = () => {
    switch (selectedTab) {
      case 'profile':
        return (
          <div className="text-center">
            <img
              src={
                profileData.profilePhoto instanceof File
                  ? URL.createObjectURL(profileData.profilePhoto)
                  : profileData.profilePhoto || '/default-avatar.png'
              }
              alt="Profile"
              className="h-24 w-24 rounded-full mx-auto border-4
border-gray-200 mb-4"
            />
            {isEditing ? (
              <div className="mt-4">
                <input
                  type="text"
                  name="name"
                  value={profileData.name}
                  onChange={handleInputChange}
                  placeholder="Name"
                  className="border border-gray-300 rounded-lg p-2 w-full
mt-2"
                />
                <input
                  type="text"
                  name="country"
                  value={profileData.country}
                  onChange={handleInputChange}
                  placeholder="Country"
                  className="border border-gray-300 rounded-lg p-2 w-full
mt-2"
                />
                <input
                  type="text"
                  name="favoriteBook"
                  value={profileData.favoriteBook}
                  onChange={handleInputChange}
                  placeholder="Favorite Book"
                  className="border border-gray-300 rounded-lg p-2 w-full
mt-2"
                />
                <input
                  type="file"
                  accept="image/*"
                  onChange={handlePhotoChange}
                  className="border border-gray-300 rounded-lg p-2 w-full
mt-2"
                />
                {loading ? (
                  <p>Saving...</p>
                ) : (
                  <button onClick={handleSave} className="bg-dark-green
text-white px-6 py-2 rounded-lg mt-4">
                    Save
                  </button>
                )}
                {error && <p className="text-red-500 mt-2">{error}</p>}
              </div>
            ) : (
              <div>
                <p className="font-semibold text-lg">Name:
{profileData.name}</p>
                <p className="font-semibold text-lg">Email:
{profileData.email}</p>
```

```
                <p className="font-semibold text-lg">Country:
{profileData.country}</p>
                <p className="font-semibold text-lg">Favorite Book:
{profileData.favoriteBook}</p>
                <button onClick={() => setIsEditing(true)}
className="bg-dark-green text-white px-6 py-2 rounded-lg mt-4">
                  Edit
                </button>
              </div>
            )}
          </div>
        );
      case 'favorites':
        return (
          <div className="p-4">
    <h3 className="text-2xl font-semibold mb-4">Favorite Books</h3>
    {favorites.length > 0 ? (
      <ul>
        {favorites.map((book, index) => (
          <li key={index} className="mb-4">
            <img
              src={book.coverImageUrl}
              alt={book.title}
              className="w-16 h-24 object-cover inline-block mr-4"
            />
            <span className="font-bold">{book.title}</span> by
{book.author}
          </li>
        ))}
      </ul>
    ) : (
      <p>No favorite books yet.</p>
    )}
  </div>
);
      case 'downloads':
        return (
          <div className="p-4">
            {downloads.length > 0 ? (
              <ul>
```

```
                {downloads.map((download, index) => (
                  <li key={index} className="mb-2">
                    <span className="font-bold">{download.title}</span> -
Downloaded on{' '}
                    {new Date(download.downloadedAt).toLocaleDateString()}
                  </li>
                ))}
              </ul>
            ) : (
              <p>No downloads yet.</p>
            )}
          </div>
        );
      default:
        return null;
    }
  };

  return (
    <div className="min-h-screen flex bg-gray-100">
      {/* Sidebar */}
      <div className="w-1/4 bg-dark-green text-white p-6 rounded-l-lg">
        <div className="text-center mb-6">
          <img
            src={
              profileData.profilePhoto instanceof File
                ? URL.createObjectURL(profileData.profilePhoto)
                : profileData.profilePhoto || '/default-avatar.png'
            }
            alt="Profile"
            className="h-24 w-24 rounded-full mx-auto border-4
border-gray-200"
          />
          <h2 className="mt-4 font-semibold">User Profile</h2>
        </div>
        <nav className="space-y-4">
          <button
            onClick={() => setSelectedTab('profile')}
            className={`w-full text-left py-2 px-4 rounded ${
              selectedTab === 'profile' ? 'bg-gray-700' : 'bg-dark-green'
            }`}
          >
            Profile
          </button>
          <button
            onClick={() => setSelectedTab('favorites')}
            className={`w-full text-left py-2 px-4 rounded ${
              selectedTab === 'favorites' ? 'bg-gray-700' :
'bg-dark-green'
            }`}
          >
            Favorites
          </button>
          <button
            onClick={() => setSelectedTab('downloads')}
            className={`w-full text-left py-2 px-4 rounded ${
              selectedTab === 'downloads' ? 'bg-gray-700' :
'bg-dark-green'
            }`}
          >
            Downloads
          </button>
        </nav>
      </div>

      {/* Main Content */}
      <div className="flex-grow bg-white p-6
rounded-r-lg">{renderContent()}</div>
    </div>
  );
}
```

When click heart icon , there is a error

```
 GET /api/auth/session 200 in 4586ms
 ○ Compiling /api/user/favorite ...
 GET /categories/Fiction/66dc7b030ae746ded871987c?_rsc=foluj
200 in 7069ms
 ✓ Compiled /api/books/[id] in 6.2s (743 modules)
 ✓ Compiled in 1ms (790 modules)
 ✓ Compiled in 1ms (790 modules)
 ✓ Compiled in 2ms (790 modules)
Already connected to MongoDB.
Already connected to MongoDB.
 GET /api/user/favorite?email=hirannikka@gmail.com 200 in 12301ms
Failed to add book to favorites: Error: User validation failed: favorites.0.bookId: Path `bookId` is
required., favorites.1.bookId: Path `bookId` is required., favorites.2.bookId: Path `bookId` is
required.
    at ValidationError.inspect (D:\text
library\library\web\node_modules\mongoose\lib\error\validation.js:50:26)
    at formatValue (node:internal/util/inspect:805:19)
    at inspect (node:internal/util/inspect:364:10)
    at formatWithOptionsInternal (node:internal/util/inspect:2298:40)
    at formatWithOptions (node:internal/util/inspect:2160:10)    at console.value
(node:internal/console/constructor:351:14)
    at console.warn (node:internal/console/constructor:384:61)
    at console.error (D:\text
library\library\web\node_modules\next\dist\client\components\react-dev-overlay\internal\helpers\
hydration-error-info.js:67:14)
    at POST (webpack-internal:///(rsc)/./src/app/api/user/favorite/route.ts:101:17)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async D:\text
library\library\web\node_modules\next\dist\compiled\next-server\app-route.runtime.dev.js:6:5503
8
    at async ek.execute (D:\text
library\library\web\node_modules\next\dist\compiled\next-server\app-route.runtime.dev.js:6:4580
8)
    at async ek.handle (D:\text
library\library\web\node_modules\next\dist\compiled\next-server\app-route.runtime.dev.js:6:5629
2)
    at async doRender (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:1377:42)
    at async cacheEntry.responseCache.get.routeKind (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:1599:28)
    at async DevServer.renderToResponseWithComponentsImpl (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:1507:28)
    at async DevServer.renderPageComponent (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:1931:24)
    at async DevServer.renderToResponseImpl (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:1969:32)
    at async DevServer.pipeImpl (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:920:25)
```

```
    at async NextNodeServer.handleCatchallRenderRequest (D:\text
library\library\web\node_modules\next\dist\server\next-server.js:272:17)
    at async DevServer.handleRequestImpl (D:\text
library\library\web\node_modules\next\dist\server\base-server.js:816:17)    at async D:\text
library\library\web\node_modules\next\dist\server\dev\next-dev-server.js:339:20
    at async Span.traceAsyncFn (D:\text
library\library\web\node_modules\next\dist\trace\trace.js:154:20)
    at async DevServer.handleRequest (D:\text
library\library\web\node_modules\next\dist\server\dev\next-dev-server.js:336:24)
    at async invokeRender (D:\text
library\library\web\node_modules\next\dist\server\lib\router-server.js:174:21)
    at async handleRequest (D:\text
library\library\web\node_modules\next\dist\server\lib\router-server.js:353:24)
    at async requestHandlerImpl (D:\text
library\library\web\node_modules\next\dist\server\lib\router-server.js:377:13)
    at async Server.requestListener (D:\text
library\library\web\node_modules\next\dist\server\lib\start-server.js:141:13) {
  errors: {
    'favorites.0.bookId': ValidatorError: Path `bookId` is required.
        at validate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1385:13)
        at SchemaType.doValidate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1369:7)
        at D:\text library\library\web\node_modules\mongoose\lib\document.js:3071:18
        at process.processTicksAndRejections (node:internal/process/task_queues:77:11) {
      properties: [Object],
      kind: 'required',
      path: 'bookId',
      value: undefined,
      reason: undefined,
      [Symbol(mongoose#validatorError)]: true
    },
    'favorites.1.bookId': ValidatorError: Path `bookId` is required.
        at validate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1385:13)
        at SchemaType.doValidate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1369:7)
        at D:\text library\library\web\node_modules\mongoose\lib\document.js:3071:18
        at process.processTicksAndRejections (node:internal/process/task_queues:77:11) {
      properties: [Object],
      kind: 'required',
      path: 'bookId',
      value: undefined,
      reason: undefined,
      [Symbol(mongoose#validatorError)]: true
    },
    'favorites.2.bookId': ValidatorError: Path `bookId` is required.
        at validate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1385:13)
        at SchemaType.doValidate (D:\text
library\library\web\node_modules\mongoose\lib\schemaType.js:1369:7)
        at D:\text library\library\web\node_modules\mongoose\lib\document.js:3071:18
        at process.processTicksAndRejections (node:internal/process/task_queues:77:11) {
      properties: [Object],
      kind: 'required',
      path: 'bookId',
      value: undefined,
      reason: undefined,
      [Symbol(mongoose#validatorError)]: true
    }
  },
  _message: 'User validation failed'
}
 POST /api/user/favorite 500 in 12804ms
```

Do my task. To do that update codes. If codes want update