E-Commerce Project

# Functionality of the E-Commerce web site

This document is a summary of my e-commerce site, named ‘Ear Candy’ as it provides audiobooks for users to select for purchasing. This document provides an overview of its components and its functionality.

**Table of Contents**

[**Functionality of the E-Commerce web site 1**](#_g1bo2mq1ox6w)

[Using the application - What the web app looks like and what it does 2](#_h4ihkg4yucm7)

[Main Page 2](#_3usynv34jyro)

[The Main Page Image 2](#_doa2972cmzc5)

[The Information Page 3](#_flq8o525pjew)

[Information Page Image 3](#_rv4proyno0uo)

[The Basket 3](#_4vquywtuetc4)

[The Basket Image 4](#_2qtv9gv0tbi6)

[What the code does 5](#_ih4jajjtzhgs)

[App.js 5](#_fvq000h4l351)

[Imports 5](#_kxx2nixz9gef)

[App.js 5](#_pe5bobk8htdf)

[State Management 5](#_97mdywtqp45s)

[Functions: 6](#_7cne3g6gjitz)

[addAudiobookToBasket: 6](#_xml0n3oaejlb)

[handleAudiobookClick: 6](#_ifp0tc51pe3o)

[handleCloseAudiobookPage: 6](#_l6pugxg3za27)

[scrollToTop : 6](#_x565vrqjthgj)

[Header and Main Content: 7](#_wge7vmynbw1j)

[Rendering the main content area where audiobooks are displayed. 7](#_st8m9lwr0qui)

[Audiobook Listing and Detailed View: 8](#_7251u2ttfew3)

[Basket Section: 8](#_umvcx69mou8u)

[Effect Handling 9](#_9t06dsco0ehb)

[Event Handling 9](#_r0hq1pomqiuy)

[handleAudioBookClick : 9](#_8a0ju6lct8s6)

[selectAudioBook: 9](#_384mq48g06l5)

[handleCloseAudiobookPage: 9](#_k6bgseolx503)

[scrollToTop Function: 9](#_uanzya7qc2kp)

[Component Structure 10](#_t9rwmvp1vfwc)

[Header: 10](#_m5c7xfnsplx9)

[AudiobookPage: 10](#_slfgu0tnl6vz)

[Rendering 10](#_f3ptttw3o7w)

[API 11](#_1euodyiu5vq0)

[AudioBook Database 12](#_la3o7ndjhsvc)

[SQL 12](#_qiury7rz7vm2)

[Testing 13](#_4ng7or2tf55i)

## Using the application - What the web app looks like and what it does

The application is opened in a web browser.

On loading the user is presented with the main page, which has the full list of audiobooks available in the database.

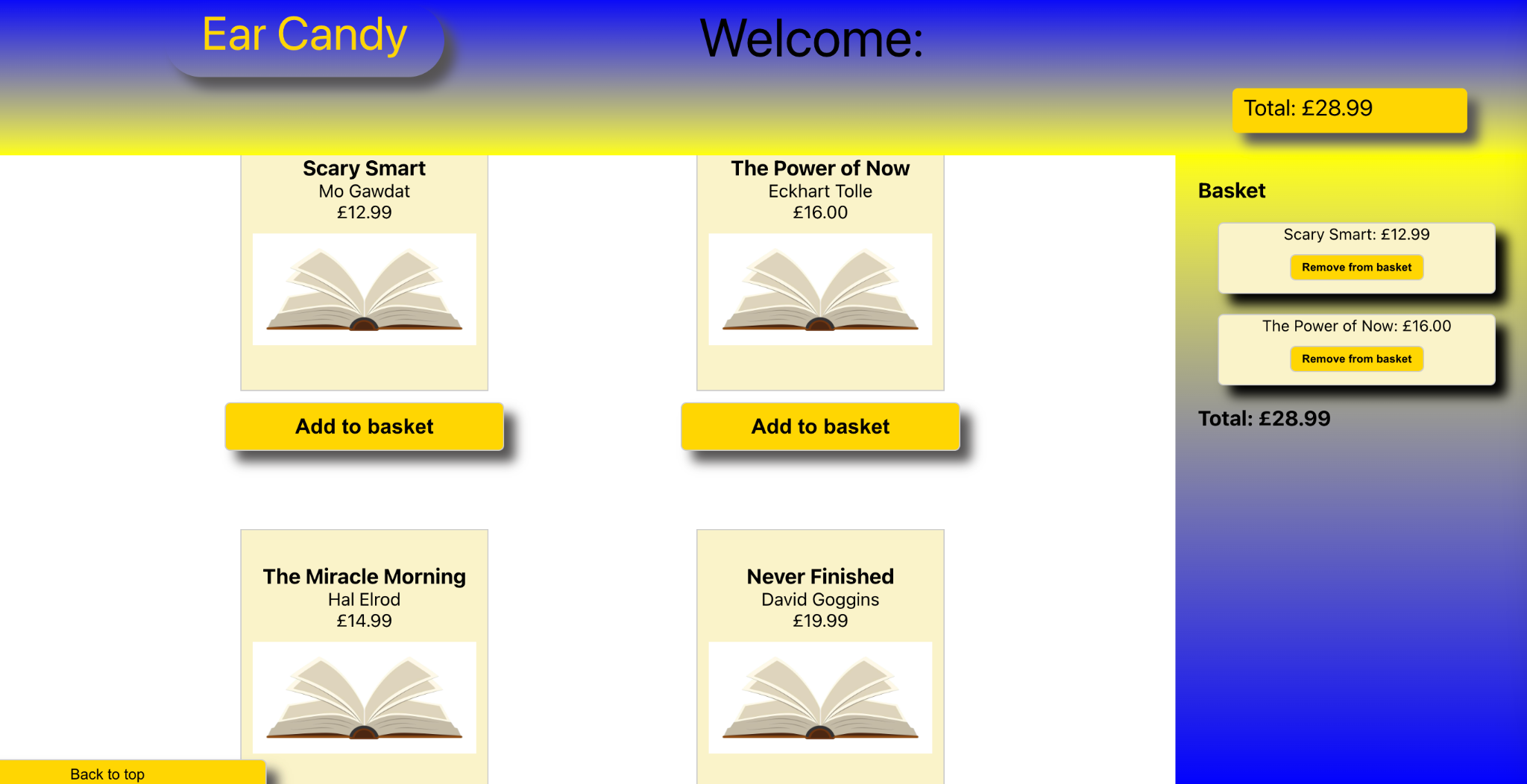
The web page consists of:

* Header - The user can view the header that only contains the name “Ear Candy” , welcome and a total in £s currently, but could contain other details including username and log in/out functionality.
* Main content - The main page renders and displays the audiobooks requested from the database, onto the screen. Displayed in two columns, showing two audiobooks on each row. All the audiobooks from the database are received and can be viewed by scrolling down the page.
* Basket Side-bar - The ‘Basket’ in the right side panel of the web page displays the audiobooks that have been added to the basket Each audiobook added appears as a separate box containing the audiobook title, the cost, and a button to remove the audiobook from the basket. Below the audiobooks is the ‘Total’ cost of all the audiobooks added.
* Scroll to top button - At the bottom left side of the page is a ‘Back to top’ button. This can be clicked on by the user to enable a quick return to the top of the page and does not require the user to scroll all the way back.

### Main Page

With the main page loaded the user can view all the audiobooks for purchase. Each book is displayed with the title, author, price, and an image.

#### The Main Page Image



### The Information Page

Viewing audiobook information:

From the main page, the user has the ability to view information on each of the audiobooks by clicking on the audiobook container. When a user hovers their cursor over an audiobook the cursor will change to a pointer, providing a visual clue that a user can click and something will happen. When a user does click on one of the audiobooks this action will display additional information about the audiobook, including its title, author, book length in hours and minutes, language, summary, and price.

#### Information Page Image



### The Basket

The user has the ability to add the book to their basket, using the ‘Add to basket’ button, or return to the home page using the ‘Return to Home Page’ button.

Adding/removing audiobooks and viewing the basket:

The user has the option to add a book to the basket for purchase using the ‘Add to basket button’. This can be done from the home/main page or the information page. The user can add any of the books to the basket and these will be displayed in the right-hand side panel. The side-panel displays the audiobook selected, including the title and price. There is a button included to remove the book from the basket. The user is also able to view the total cost of the audiobooks they have selected. This is a running total of all the books selected and is updated to reflect each addition or when an audiobook is removed.

#### The Basket Image



# 

## 

## What the code does

The ‘Ear Candy’ app consists of the following:

* e-commerce-2 folder containing all the files required to run the app.
* e-commerce-app containing the relevant files for the app to load and request the data from the database.
  + App.js - main entry component
  + AudioBook.jsx - contains Audiobook class and audiobook instances
  + AudiobookPage.jsx - renders audiobook information page
  + Header.jsx - renders header
  + App.css - styles sheet
  + App.test.js - testing for correct rendering of audiobooks
  + e-commerce-API.js - used to request data from API
* e-commerce-API containing index.js, used to connect and fetch data the database holding the database holding the Audiobook tables.

### App.js

App.js serves as the main entry point of the audiobook e-commerce application. It imports necessary components, manages application state, and handles user interactions. Below is a summary of its key features and functionalities:

#### Imports

App.js imports the necessary CSS files, components, and API functions required for the application.

##### App.js

import "./App.css";

import { audioBookList } from "./AudioBook";

import Header from "./Components/Header";

import React, { useState, useEffect } from "react";

import AudiobookPage from "./Components/AudiobookPage";

import BookImage from "./Components/open-book-2.jpeg";

import { getAudiobooks } from "./e-commerce-API";

#### State Management

The App.js component utilises the useState hook. It initialises the following state variables to use for state management:

* basket: the basket and setBasket tracks audiobooks added or removed to manage the state of the basket, to allow for adding and removing audiobooks.
* total: total and setTotal keeps track of the total cost, managing the state of the total cost when audiobooks are added or removed from the basket.
* selectAudioBook: selectAudioBook and setSelectAudioBook stores and represents the currently selected audiobook to allow for display of all the audiobooks or the information for one audiobook that has been selected.

#### Functions:

#### addAudiobookToBasket:

Function is defined to add an audiobook to the basket and update the total cost.

*// function to add audiobook to basket and update the total cost*

const addAudiobookToBasket = (audioBook) => {

*//add the audiobook to the basket*

setBasket([...basket, audioBook]);

*//update the total cost*

setTotal(total + audioBook.price);

};

#### handleAudiobookClick:

Function to handle the click event when an audiobook is clicked for detailed viewing.

*// function to handle the click event is an audiobook is clicked*

const handleAudiobookClick = (audioBook) => {

setSelectedAudiobook(audioBook);

console.log( 'Audiobook clicked" ${audioBook.title} ${audioBook.author}`);

};

#### handleCloseAudiobookPage:

Function to handle the close event of the audiobook page.

*// function to handle the close event of the audiobook page*

const handleCloseAudiobookPage = () => {

setSelectedAudiobook(null);

};

#### scrollToTop :

Function to scroll the page to the top. Pages all include a button to scroll the page to the top for user convenience.

*// setup scroll to top button*

const scrollToTop = () => {

window.scrollTo({ top: 20, left: 20, behavior: "smooth" });

};

#### Header and Main Content:

Displays the application header, including the logo, user welcome message, and total spend summary.

<div className="App-container">

{*/\* display the header \*/*}

<Header

headerItems={[

{ className: "header-logo-box", text: "Ear Candy" },

{

className: "headerhandle-username-box",

text: "Welcome: ",

},

{ className: "header-welcome-box", text: "" },

{

className: "header-spend-summary-box",

text: "Total: £" + total.toFixed(2),

},

]}

/>

#### Rendering the main content area where audiobooks are displayed.

- Uses conditional rendering to display either the detailed view of a selected audiobook or the list of available audiobooks.

{*/\* display the main content of audiobooks here \*/*}

<div className="main-page-content">

{*/\* using ternary check if audiobook is selected \*/*}

{selectedAudiobook ? (

*// if audiobook is not selected, display the audiobook page*

<div className="books-container">

<AudiobookPage

addAudiobookToBasket={addAudiobookToBasket}

audioBook={selectedAudiobook}

onClose={handleCloseAudiobookPage}

/>

</div>

) : (

*// if audiobook is selected, display the audiobook list*

<div className="books-container">

{*/\* map through the audiobook list and display each audiobook \*/*}

{audioBookList.map((audioBook, index) => (

<div className="audiobook-wrapper" key={index}>

<div

className="audiobook-container"

*// pass the audiobook object to the handleAudiobookClick function*

onClick={() => handleAudiobookClick(audioBook)}

>

{*/\* display audiobook details and image \*/*}

<h3>{audioBook.title}</h3>

<p>{audioBook.author}</p>

<p>£{audioBook.price.toFixed(2)}</p>

<img src={BookImage} alt="book" />

</div>

</div>

))}

</div>

#### Audiobook Listing and Detailed View:

- Lists audiobooks available for purchase, each displaying title, author, price, and an image.

- Allows users to click on an audiobook to view detailed information.

- Provides options to add audiobooks to the basket directly from the list or detailed view.

#### Basket Section:

- When the “Add to basket” button is clicked, it updates the basket with the addAudioBookToBasket function. Displays the current contents of the user's basket, including audiobook titles, prices, and remove buttons.

- Dynamically updates the total cost based on the contents of the basket.

{*/\* button to pass audiobook object directly to basket \*/*}

<button

className="add-audiobook-button"

onClick={() => addAudiobookToBasket(audioBook)}

>

<h3>Add to basket</h3>

</button>

Overall, App.js orchestrates the user interface and functionality of the audiobook e-commerce application, providing users with an intuitive browsing and purchasing experience.

### Effect Handling

The effect handling is completed using the useEffect hook.

Inside the useEffect function 'fetchAudioBooks()’ is called. This function is responsible for fetching audio books asynchronously from an external data source (e.g., e-commerce-API). Once the data is fetched, it updates the state variable ‘audioBookList’ using ‘setAudioBookList(audioBooks)’.

The effect function is asynchronous, meaning it waits for the ‘fetchAudioBooks()’ function to complete its execution before finishing its own execution.

The effect ensures that the audio books are fetched and available for rendering when the component mounts for the first time.

This code is used to trigger the fetching of audio books from the external API, ensuring that the ‘audioBookList’ state variable is populated with the latest data.

### Event Handling

#### **handleAudioBookClick** :

This function is responsible for handling the click event when an audiobook is clicked. It takes an audiobook object as an argument and updates the ‘selectedAudiobook’ state with the clicked audiobook, triggering the display of detailed information about that audiobook.

#### **selectAudioBook**:

This function is similar to ‘handleAudioBookClick’ and is likely a typo or a duplicate function name. Assuming it serves the same purpose, it would also be responsible for selecting an audiobook when clicked, updating the 'selectedAudiobook ' state accordingly.

#### **handleCloseAudiobookPage**:

This function is used to handle the close event of the audiobook page. When called, it resets the 'selectedAudiobook' state to 'null' , effectively closing the detailed display of the audiobook page. This function is essential for managing the visibility of the audiobook page and returning to the main content view.

#### **scrollToTop Function**:

This implements a function to scroll the page to the top of the page when called, to allow the user to navigate to the top of the page with the ease of a click of a button.

### Component Structure

#### Header:

The Header component renders the header of the e-commerce project. It receives 'headerItems' as a prop, which is an array containing objects representing different header elements. The component maps over 'headerItems' , rendering each element with its corresponding CSS class and text content. The 'key' attribute is assigned to each element to ensure proper rendering and efficient updates. Finally, the header elements are enclosed within a 'header' tag with the class name "header-container". This component facilitates the dynamic rendering of header elements based on the provided props.

#### AudiobookPage:

The `AudiobookPage ' component is responsible for rendering detailed information about an audiobook to the screen. It receives 'audioBook ', 'addAudiobookToBasket ', and 'onClose' as props. The audiobook information is displayed, including its title, author, book length, language, summary, and price. The component also renders two buttons: one for adding the audiobook to the basket and another for returning to the home page. When the "Add to basket" button is clicked, it invokes the 'addAudiobookToBasket' function passed via props, which adds the audiobook to the basket. Similarly, when the "Return to Home Page" button is clicked, it triggers the 'onClose' function to return to the home page. The component utilises an image component ('BookImage') for displaying the book cover.

### Rendering

Conditional rendering in my React project is based on the 'selectedAudiobook ' state variable:

Ternary Operator :

- The ternary operator ( 'selectedAudiobook ? (...) : (...) ') is used to conditionally render JSX based on whether an audiobook is selected or not.

- If 'selectedAudiobook ' is truthy (i.e., an audiobook is selected), the first part of the ternary operator is executed. It renders the 'AudiobookPage ' component, displaying detailed information about the selected audiobook.

- If 'selectedAudiobook ' is falsy (i.e., no audiobook is selected), the second part of the ternary operator is executed. It renders a list of audiobooks available for selection.

Rendering Audiobook Page :

- If an audiobook is selected ( 'selectedAudiobook ' is truthy), the 'AudiobookPage ' component is rendered inside a '<div> ' with the class name "books-container".

- The 'AudiobookPage ' component is passed props such as 'addAudiobookToBasket ' for adding the audiobook to the basket, 'audioBook ' to display detailed information about the selected audiobook, and 'onClose ' to handle closing the audiobook page.

Rendering Audiobook List :

- If no audiobook is selected ( 'selectedAudiobook ' is falsy), a list of audiobooks is rendered inside a '<div> ' with the class name "books-container".

- Each audiobook in the 'audioBookList ' is mapped and rendered within a '<div> ' with the class name "audiobook-wrapper".

- The audiobook details, including title, author, price, and image, are displayed within a '<div> ' with the class name "audiobook-container".

- Each audiobook also includes an "Add to basket" button, which allows users to add the audiobook to their basket.

This approach provides a dynamic user interface where users can switch between viewing detailed information about a selected audiobook and browsing through all the available audiobooks.

### API

To create an API for this project and to render the audiobooks from an Azure server using SQL requests:

The required dependencies, including Express and Cross-Origin Resource Sharing (cors), were installed into the e-commerce-API project folder ( 'express ' and 'cors '). The 'mssql ' package ( 'npm install mssql ') was installed for connecting to the SQL Server.

These services were defined at the top of the API file to be used.

An SQL connection was configured to the Azure server.

A route was created in Express to handle audiobook requests.

An SQL query was included to retrieve audiobook data from the Azure SQL database.

The audiobook data is sent as a response from the Express route.

Import necessary modules

*// this file is used to fetch data from the API*

const express = require("express");

const sql = require("mssql");

const port = 3001;

const cors = require("cors");

Initialize Express app

const app = express();

app.use(express.json());

Use CORS middleware

app.use(cors());

Configuration of Azure SQL Server with credentials and database details.

const sqlConfig = { #configuration details here

user: “your\_username”,

password: “your\_password,

server: “”your\_server\_name.database.windows.net”,

database: “your\_database\_name”

options: {

Encrypt: true // to use encryption

}

};

*// using the GET method to access the url*

app.get("/", (req, res) => {

res.send("Default found");

});

Define route to render audiobooks

*// using the GET method to access the AudioBook and return the data to the screen*

app.get("/Audiobook", async (req, res) => {

app.get('/audiobooks', async (req, res) => {

try {

let pool = await sql.connect(sqlConfig);

// Query to retrieve audiobook data

let audiobooks = await pool.request().query("SELECT \* FROM Audiobook");

Send the audiobook data as response

res.json(audiobooks.recordset);

} catch (error) {

res.status(500);

res.send(error.message);

}

With this setup, when you make a GET request to '/audiobooks ', the server will fetch audiobook data from the Azure SQL Server and send it as a JSON response.

### AudioBook Database

#### SQL

The AudioBook Tables were created using SQL.   
SQL example to create table

CREATE TABLE AudioBook(

audioBook\_id INT IDENTITY(1, 1) PRIMARY KEY,

title VARCHAR (100),

author VARCHAR (50),

price FLOAT (2),

bookLength FLOAT (2),

language VARCHAR (50),

summary VARCHAR (250),

image VARCHAR (250)

);

INSERT INTO AudioBook(audioBook\_id, title, author, price, bookLength, language, summary, image) VALUES (1, 'Atomic Habits','James Clear',9.99,5.35,'English','An Easy and Proven Way to Build Good Habits and Break Bad Ones', 'BookImage');

SELECT \* FROM AudioBook;

### Testing

To ensure the proper functioning of the audiobook request action, a series of tests were conducted. The first step involved importing the necessary modules essential for running the test suite.

Subsequently, an instance of an audiobook object was created within the test file. This instance served as the basis for comparison against the expected data structure retrieved from the API.

Finally, thorough examination was performed to validate that the audiobook instance possesses the correct properties, ensuring conformity with the expected data format.

Upon successful completion of the initial test, a deliberate alteration was made to the audiobook data to induce a failure. Specifically, the language attribute was modified from "English" to "German", diverging from the expected value.

Both the successful and failed test outcomes were documented and are visually represented in the accompanying screenshots below, providing comprehensive insight into the efficacy of the testing process.

