Version 0.1

**Otc 29, 2021**

Online Bookstore

SYSTEM

REQUIREMENT

**SPECIFICATION**

**Author:** Dao Minh Hieu

**Security Classification:**

110 Bishopsgate, London EC2N 4AY

Tel: +44 (0)20 7333 0033

Email: [info@nashtechglobal.com](mailto:info@nashtechglobal.com)



Table of contents

[1. Overview 2](#_Toc86586372)

[**1.1** Purpose of this document 2](#_Toc86586373)

[2. Functional requirements 2](#_Toc86586374)

[**2.1** Login/logout feature 2](#_Toc86586375)

[2.1.1 Description 2](#_Toc86586376)

[2.1.2 Techniques 2](#_Toc86586377)

[**2.2** Register page 2](#_Toc86586378)

[2.2.1 Description 2](#_Toc86586379)

[2.2.2 Techniques 2](#_Toc86586380)

[**2.3** Get/Update Customer 3](#_Toc86586381)

[2.3.1 Description 3](#_Toc86586382)

[2.3.2 Techniques 3](#_Toc86586383)

[**2.4** Book Dashboard 3](#_Toc86586384)

[2.4.1 Description 3](#_Toc86586385)

[2.4.2 Techniques 3](#_Toc86586386)

[**2.5** Catalog Menu 4](#_Toc86586387)

[2.5.1 Description 4](#_Toc86586388)

[2.5.2 Techniques 4](#_Toc86586389)

[**2.6** Book Catalog Dashboard 4](#_Toc86586390)

[2.6.1 Description 4](#_Toc86586391)

[2.6.2 Techniques 4](#_Toc86586392)

[**2.7** Pagination 5](#_Toc86586393)

[2.7.1 Description 5](#_Toc86586394)

[2.7.2 Techniques 5](#_Toc86586395)

[**2.8** Product Detail 5](#_Toc86586396)

[2.8.1 Description 5](#_Toc86586397)

[2.8.2 Techniques 5](#_Toc86586398)

[**2.9** Rating Stars 6](#_Toc86586399)

[2.9.1 Description 6](#_Toc86586400)

[2.9.2 Techniques 6](#_Toc86586401)

[**2.10** Comment 6](#_Toc86586402)

[2.10.1 Description 6](#_Toc86586403)

[2.10.2 Techniques 6](#_Toc86586404)

[**2.11** Preview feature 7](#_Toc86586405)

[2.11.1 Description 7](#_Toc86586406)

[2.11.2 Techniques 7](#_Toc86586407)

[**2.12** Shopping Cart 7](#_Toc86586408)

[2.12.1 Description 7](#_Toc86586409)

[2.12.2 Techniques 8](#_Toc86586410)

[**2.13** Historical Order 8](#_Toc86586411)

[2.13.1 Description 8](#_Toc86586412)

[2.13.2 Techniques 8](#_Toc86586413)

[**2.14** Error boundary 8](#_Toc86586414)

[2.14.1 Description 8](#_Toc86586415)

[2.14.2 Techniques 8](#_Toc86586416)

[**2.15** Search feature 9](#_Toc86586417)

[2.15.1 Description 9](#_Toc86586418)

[2.15.2 Techniques 9](#_Toc86586419)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document Version History | | | | | |
| **Version** | **Effective Date** | **Author** | **Details** | **Reviewer** | **Approvers** |
| 0.1 | Otc 29 2021 | Dao Minh Hieu | Created | All members | Le Cao |

Confidentiality

This document is distributed on a restricted basis, is commercial in confidence to the recipient, and may not be used for any purpose other than that associated with a NashTech project. The contents of this document may not be disclosed to any third parties without the expressed advance written authorisation of NashTech.

# Overview

## Purpose of this document

This document provides the functional requirements covered in the project “Online Bookstore”. All the features will include a feature name, a short description and React techniques that the developer are applying to their works.

# Functional requirements

Describe some main functional requirements and how the solution has covered each of the items.

## Login/logout feature

### Description

Allow user can login/log out/register via IdentityServer4.

### Techniques

* + **Functional Component**
    - Practise control functional component and manage state with hooks.
  + **Redux**
    - React-tool-kit
      * Create reducer-action in redux with tool-kits.
    - React-redux
      * Store data to global state.
      * Get data from a global state with react-redux.
  + **OAuth-2.0**
    - Oidc-clients
      * Connect React projects with IdentityServer4.
      * Configure setting in React-project.
      * Configure setting in IdentityServer4.

## Register page

### Description

* This page will help users register a new account.

### Techniques

* + IdentityServer4 .Net Core 3.1, SQL Server.

Graphical user interface, application

Description automatically generated

## Get/Update Customer

### Description

* After registering an account, the user should be updated the personal info to purchase products on the website.
* To update info, the user must be logged in. After that, the user reaches to “My Account” page where has some text fields (first name, last name, phone number…). All of them are required, so the user must input valid values to them and press the “update” button to update the info.

### Techniques

* Use functional components to create the MyAccount page.
* Using State Hook, Effect Hook to update state and get customer data.
* In the MyAccount page, userId is used to get data from CustomerService and this can be retrieved from the Redux store (loginUser) where useSelector must be used.

## Book Dashboard

### Description

* + Show list products in the system.

### Techniques

* + **Functional Component**
    - Simple approach since no need to handle special lifecycle
  + **Hook**
    - useEffect: Add side effect fetching list product and handle component’s lifecycle
    - useContext: manage global state.
    - useReducer: manage logic state.
  + **Library**:
    - semantic-UI-react: design UI/UX.(button, link, gridView,..).

## Catalog Menu

### Description

* Load the list catalogue for the menu from the SQL Server database. Then, users can select any catalogue to see books that belonged to that one.

### Techniques

* + Frontend: using React Hooks
    - useState: Manage local state.
    - useEffect: Add side effect fetching catalogue data.
  + Backend: Web APIs .Net Core 3.1, SQL Server.

Graphical user interface, application

Description automatically generated

## Book Catalog Dashboard

### Description

* When users click catalogue items in the navigation menu, the system will display the list of books by catalogId.

### Techniques

* + Frontend: using Redux
    - useEffect: Add side effect fetching book data.
    - useReducer: Mange states and actions to send and get book data.
    - useParams: To get query string “catalogId” to query book data.
  + Backend: Web APIs .Net Core 3.1, SQL Server.

**Graphical user interface, application

Description automatically generated**

## Pagination

### Description

* This feature divides a list of records into discrete pages. The Semantic UI React already has a component to render pagination. The component helps to handle navigating to the previous page, next page, first page, last page and select a page number through two properties: *defaultActivePage* and *totalPages*.
* Currently, the query is from the server-side. Whenever a user clicks a page number, it gets that page number and calls a query API to get a list of records.

### Techniques

When a user clicks a page number, the active page must be set as that number. To do this, I use:

* *useState* Hook with the default value of the active page is one.
* Then I call the API by using the *Axios library which helps to send a request to an outer endpoint*.

## Product Detail

### Description

* Show detail product
* Show the product’s long description.

### Techniques

* + **Functional Component**
    - Simple approach since no need to handle special lifecycle
  + **Hook**
    - useEffect: Add side effect fetching list product and handle component’s lifecycle
    - useRef: scroll to an element.
    - UseParams: get Id from link page detail that config route store
  + **Library** :
    - semantic-UI-react: design UI/UX
    - react-router-dom: handle all routing

## Rating Stars

### Description

When visiting the book detail page, users can rate if the book is good or not by stars. If a user has not logged in, the rating section will display the Rating Overview only and hide the Rating section. The Rating section has two fields: the number of stars and the comment. If the user has chosen some stars and written the comment, then the form can be submitted.

### Techniques

* I use Formik and Yup for form validation.
  + With Formik, I can initialize the default value of the fields, add validation rules which are defined by Yup and handle the form submission.
* When submitting the form, if it passes the validation, *bookId, ratingValue* and *comment* are used as parameters to call an API to update the book rating by *Axios*.
* After the book detail has been updated, the Rating Overview is updated as well. The average star rating will be recalculated, and the rating value for each star number is updated as well by using *useState* Hook.

## Comment

### Description

Giving users the ability to share their thought about a specific product. Comment’s owner is allowed to modify its content or even remove itself.

The comment feature is exposed as a standalone component, which reduces the impact on the whole codebase. Server-side pagination is also applied to reduce loading and rendering time.

### Techniques

* + **Functional Component**
    - Simple approach since no need to handle special lifecycle
  + **Context**
    - Passing through the **global state**, **props** for comment components tree
    - Passing **dispatch** down multi-level instead of callback
  + **Memoization**
    - React.memo
      * **Higher-order component**, shallow comparison **props** and **state** to prevent re-render
  + **Hook**
    - Custom Hook
      * Wrapping **useReducer** to combine both state and dispatch into one **contextType**
    - useReducer:
      * Manage complex **global state** for component tree
    - useMemo
      * Memoize **props** to use as **context** for the component tree, which will prevent re-render all consumer
    - useContext
    - useState
      * Manage **local state**
    - useCallback
      * Memoize local function using as dependencies
    - useEffect
      * Add side effect fetching comment of product
  + **Library**
    - Yup
      * Define form validation via schema
    - Formik
      * Manage form state and validation, using **render-props**
    - date-fns
      * Format date time as ago

## Preview feature

### Description

Giving users the ability to view a variation of a specific product.

### Techniques

* + **Functional Component**
    - Simple approach since no need to handle special lifecycle
  + **Hook**
    - useState
      * Manage local **state**
    - useRef
      * Preserve value between renders to prevent API request
    - useCallback
      * **Memoize** unchanged functions to use as dependencies
    - useEffect
      * Add side effect to fetch preview of product
    - React.lazy
      * Support **lazy load** component on demand
  + **Library**
    - react-zoom-pan-pinch
      * Add zoom-in capability

## Shopping Cart

### Description

This feature lets the users be able to add, remove items to a shopping cart and then do the ordering for what goods they are interested in. Moreover, they can input their information to save into the database, either increasing or decreasing the number of ordering items of a user shopping cart. Then they can click to “Check out” button to do an order.

### Techniques

* React Context: to create a common shopping cart context including cart items and common operations on the cart such as adding, removing items, getting cart item information, etc.
* useSelector: from the react-redux library to get user status whether they are logged in or not to display the “Add to cart” button conditionally.
* Conditional rendering.
* Semantic-react-UI library: a responsive library to build UI specified for React.
* Functional component: Simple approach since no need to handle special lifecycle.
* useState: Manage local **state.**
* useEffect: modify the current state.
* Axios: to create requests and send to an exposed end-point to interact with the under-system.
* Props: the input from outer components.

## Historical Order

### Description

This feature lets the users be able to retrieve historical orders to look for what they ordered and check for the receiver information.

### Techniques

* Functional component: Simple approach since no need to handle special lifecycle.
* useSelector: from the react-redux library to get user status whether they are logged in or not to display the “Add to cart” button conditionally.
* Conditional rendering.
* Semantic-react-UI library: a responsive library to build UI specified for React.
* useState: Manage local **state**
* useEffect: modify the current state.
* Axios: to create requests and send to an exposed end-point to interact with the under-system.
* Props: the input from outer components.

## Error boundary

### Description

* To handle rejected promises problem we try to add event listeners to catch unhandledRejection events in lifecycle methods (componentDidMount, componentWillUnmount) and throw an error to Error boundaries.

### Techniques

* Create Error boundary which is a class component to catch javascript errors.
* Install the error boundary as an interceptor that will encounter error requests so that we can actively handle the errors in the way that we want.
* Note: according to <https://reactjs.org/docs/error-boundaries.html>, error boundaries doesn’t catch error from
  + Event handlers.
  + Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks).
  + Server-side rendering.
  + Errors are thrown in the error boundary itself (rather than its children).

## Search feature

### Description

Allow users to search products with data retrieved from the database while allowing users to input keywords to search products.

### Techniques

* + **Functional Component**
    - Practise control functional component and manage state with hooks.
  + **Hook**
    - useRef
      * Get historical state.
    - useState
      * Manage **local state.**
    - useEffect
      * Add side effect fetching product.