HEADER.JS

import ArrowBackIcon from "@mui/icons-material/ArrowBack";

import {

Avatar,

Button,

Stack,

InputAdornment,

TextField,

} from "@mui/material";

import Box from "@mui/material/Box";

import React from "react";

import "./Header.css";

import { useHistory } from "react-router-dom";

import { Logout } from "@mui/icons-material";

import { Search, SentimentDissatisfied } from "@mui/icons-material";

const Header = ({ children, hasHiddenAuthButtons }) => {

const history = useHistory();

let showButton = (

<Button

className="explore-button"

startIcon={<ArrowBackIcon />}

variant="text"

onClick={() => {

history.push("/");

}}

>

Back to explore

</Button>

);

let showSearch = "";

if (!hasHiddenAuthButtons && children) {

showSearch = (

<TextField

className="search-desktop"

size="small"

InputProps={{

endAdornment: (

<InputAdornment position="end">

<Search color="primary" />

</InputAdornment>

),

}}

placeholder="Search for items/categories"

name="search"

value={children.search}

onChange={(e)=>children.fun(e)}

/>

);

console.log(!localStorage.getItem("token"));

if (localStorage.getItem("token")) {

showButton = (

<Stack direction={"row"} spacing={2}>

<Avatar alt={localStorage.getItem("username")} src="avatar.png" />

<p className="username-text">{localStorage.getItem("username")}</p>

<Button

variant="text"

onClick={() => {

logout();

}}

endIcon={<Logout />}

>

logout

</Button>

</Stack>

);

} else {

showButton = (

<Stack direction={{ sm: "row" }} spacing={2}>

<Button

variant="text"

onClick={() => history.push("/login", { from: "/" })}

>

Login

</Button>

<Button

className="button"

variant="contained"

onClick={() => history.push("/register", { from: "/" })}

>

Register

</Button>

</Stack>

);

}

}

const logout = () => {

localStorage.clear();

window.location.reload();

};

return (

<Box className="header">

<Box className="header-title">

<img src="logo\_light.svg" alt="QKart-icon"></img>

</Box>

{showSearch}

{showButton}

</Box>

);

};

export default Header;

HEADER.CSS

.header {

display: flex;

justify-content: space-between;

align-items: center;

padding: 1rem;

height: 5vh;

box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.2);

background: #ffffff;

}

.search-desktop {

display: none !important;

}

.username-text {

display: none;

}

.header-title {

cursor: pointer;

}

.explore-button {

color: #00a278 !important;

}

@media screen and (min-width: 768px) {

.username-text {

display: block;

margin-top: 9px !important;

}

.search-desktop {

display: block !important;

}

}

LOGIN.JS

import { Button, CircularProgress, Stack, TextField } from "@mui/material";

import { Box } from "@mui/system";

import axios from "axios";

import { useSnackbar } from "notistack";

import React, { useState } from "react";

import { useHistory, Link } from "react-router-dom";

import { config } from "../App";

import Footer from "./Footer";

import Header from "./Header";

import "./Login.css";

const Login = () => {

const { enqueueSnackbar } = useSnackbar();

const [isLoading,setIsLoading]=useState(false);

const [formData,setFormData]=useState({

username:"",

password:"",

});

const history=useHistory();

// TODO: CRIO\_TASK\_MODULE\_LOGIN - Fetch the API response

/\*\*

\* Perform the Login API call

\* @param {{ username: string, password: string }} formData

\* Object with values of username, password and confirm password user entered to register

\*

\* API endpoint - "POST /auth/login"

\*

\* Example for successful response from backend:

\* HTTP 201

\* {

\* "success": true,

\* "token": "testtoken",

\* "username": "criodo",

\* "balance": 5000

\* }

\*

\* Example for failed response from backend:

\* HTTP 400

\* {

\* "success": false,

\* "message": "Password is incorrect"

\* }

\*

\*/

const login = async (formData) => {

if (validateInput(formData)){

setIsLoading(true);

try{

let response=await axios.post(`${config.endpoint}/auth/login`,formData)

setIsLoading(false);

enqueueSnackbar("Logged in successfully",{ variant: 'success' })

if (response.status===201){

persistLogin(response.data.token,response.data.username,response.data.balance)

}

console.log(response);

history.push("/",{from:"/login"})

}catch(error){

setIsLoading(false);

let errorMsg="";

if (!error.response){

errorMsg="Something went wrong. Check that the backend is running, reachable and returns valid JSON."

}

else if (error.response.status===400){

errorMsg=`${error.response.data.message}`

}else{

errorMsg="Something went wrong. Check that the backend is running, reachable and returns valid JSON."

}

enqueueSnackbar(errorMsg ,{ variant: 'error' })

}

}

};

// TODO: CRIO\_TASK\_MODULE\_LOGIN - Validate the input

/\*\*

\* Validate the input values so that any bad or illegal values are not passed to the backend.

\*

\* @param {{ username: string, password: string }} data

\* Object with values of username, password and confirm password user entered to register

\*

\* @returns {boolean}

\* Whether validation has passed or not

\*

\* Return false and show warning message if any validation condition fails, otherwise return true.

\* (NOTE: The error messages to be shown for each of these cases, are given with them)

\* - Check that username field is not an empty value - "Username is a required field"

\* - Check that password field is not an empty value - "Password is a required field"

\*/

const validateInput = (data) => {

if (data.username===""){

enqueueSnackbar("Username is a required field",{ variant: 'error' })

return false;}

if (data.password===""){

enqueueSnackbar("password is a required field",{ variant: 'error' })

return false;}

return true;

};

// TODO: CRIO\_TASK\_MODULE\_LOGIN - Persist user's login information

/\*\*

\* Store the login information so that it can be used to identify the user in subsequent API calls

\*

\* @param {string} token

\* API token used for authentication of requests after logging in

\* @param {string} username

\* Username of the logged in user

\* @param {string} balance

\* Wallet balance amount of the logged in user

\*

\* Make use of localStorage: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

\* - `token` field in localStorage can be used to store the Oauth token

\* - `username` field in localStorage can be used to store the username that the user is logged in as

\* - `balance` field in localStorage can be used to store the balance amount in the user's wallet

\*/

const persistLogin = (token, username, balance) => {

localStorage.setItem("token", token);

localStorage.setItem("username", username);

localStorage.setItem("balance", balance);

};

function handleFormData(e){

setFormData({...formData,[e.target.name] : e.target.value});

};

return (

<Box

display="flex"

flexDirection="column"

justifyContent="space-between"

minHeight="100vh"

>

<Header hasHiddenAuthButtons />

<Box className="content">

<Stack spacing={2} className="form">

<h2 className="title">Login</h2>

<TextField

id="username"

label="Username"

variant="outlined"

title="Username"

name="username"

placeholder="Enter Username"

fullWidth

value={formData.username}

onChange={(e)=>{

handleFormData(e)

}}

/>

<TextField

id="password"

label="Password"

variant="outlined"

title="Password"

name="password"

placeholder="Enter Password"

fullWidth

type="password"

value={formData.password}

onChange={(e)=>{

handleFormData(e)

}}

/>

{isLoading?<CircularProgress className="loading"/>: <Button className="button" fullWidth variant="contained" onClick={()=> login(formData)}>

LOGIN TO QKART

</Button>}

<p className="secondary-action">

Don’t have an account?{" "}

<Link className="link" to="/register">

Register now

</Link>

</p>

</Stack>

</Box>

<Footer />

</Box>

);

};

export default Login;

LOGIN.CSS

.content {

background-image: url("../assets/hero\_image.png");

background-position: center;

background-repeat: no-repeat;

background-size: cover;

display: flex;

justify-content: center;

height: 70vh;

}

.form {

background-color: white;

border-radius: 10px;

padding: 2rem;

min-width: 80%;

margin: 2rem;

box-shadow: 0px 0px 10px 0px rgba(0, 0, 0, 0.2);

}

.title {

color: #00a278;

}

.secondary-action {

padding: 1rem 0;

}

.link {

font-weight: 700;

text-decoration: none;

}

.link:focus {

color: #00a278;

}

.link:active {

color: #45c09f;

}

.link:visited {

color: #45c09f;

}

@media screen and (min-width: 768px) {

.content {

justify-content: flex-end;

}

.form {

min-width: 24rem;

}

}

REGISTER.JS

import { AlignHorizontalCenter } from "@mui/icons-material";

import { Button, CircularProgress, Stack, TextField } from "@mui/material";

import { Box } from "@mui/system";

import axios from "axios";

import { useSnackbar } from "notistack";

import React, { useState } from "react";

import { config } from "../App";

import Footer from "./Footer";

import Header from "./Header";

import "./Register.css";

import {

Link,useHistory

} from "react-router-dom";

const Register = () => {

const { enqueueSnackbar } = useSnackbar();

const [formData,setFormData]=useState({

username:"",

password:"",

confirmPassword:""

});

const [isLoading,setIsLoading]=useState(false);

const history=useHistory();

// TODO: CRIO\_TASK\_MODULE\_REGISTER - Implement the register function

/\*\*

\* Definition for register handler

\* - Function to be called when the user clicks on the register button or submits the register form

\*

\*

\* @param {{ username: string, password: string, confirmPassword: string }} formData

\* Object with values of username, password and confirm password user entered to register

\*

\* API endpoint - "POST /auth/register"

\*

\* Example for successful response from backend for the API call:

\* HTTP 201

\* {

\* "success": true,

\* }

\*

\* Example for failed response from backend for the API call:

\* HTTP 400

\* {

\* "success": false,

\* "message": "Username is already taken"

\* }

\*/

const register = async (formData) => {

if (validateInput(formData)){

setIsLoading(true);

let data={ username:formData.username,

password:formData.password}

try{

let response= await axios.post(`${config.endpoint}/auth/register`,data);

console.log(response,"888");

setIsLoading(false);

enqueueSnackbar("Registered successfully",{ variant: 'success' })

history.push("/login",{from:"/register"})

}

catch(error){

setIsLoading(false);

let errorMsg="";

if (error.response&&error.response.status===400){

errorMsg=`${error.response.data.message}`

}else{

errorMsg="Something went wrong. Check that the backend is running, reachable and returns valid JSON."

}

enqueueSnackbar(errorMsg ,{ variant: 'error' })

}

}

};

// TODO: CRIO\_TASK\_MODULE\_REGISTER - Implement user input validation logic

/\*\*

\* Validate the input values so that any bad or illegal values are not passed to the backend.

\*

\* @param {{ username: string, password: string, confirmPassword: string }} data

\*

\* Object with values of username, password and confirm password user entered to register

\*

\* @returns {boolean}

\* Whether validation has passed or not

\*

\* Return false if any validation condition fails, otherwise return true.

\* (NOTE: The error messages to be shown for each of these cases, are given with them)

\* - Check that username field is not an empty value - "Username is a required field"

\* - Check that username field is not less than 6 characters in length - "Username must be at least 6 characters"

\* - Check that password field is not an empty value - "Password is a required field"

\* - Check that password field is not less than 6 characters in length - "Password must be at least 6 characters"

\* - Check that confirmPassword field has the same value as password field - Passwords do not match

\*

\*/

const validateInput = (data) => {

if (data.username.length<6){

if (data.username===""){

enqueueSnackbar("Username is a required field",{ variant: 'error' })

return false;

}else{

enqueueSnackbar("Username must be at least 6 characters",{ variant: 'error' })

return false;

}

}

if (data.password.length<6){

if (data.password===""){

enqueueSnackbar("password is a required field",{ variant: 'error' })

return false;

}else{

enqueueSnackbar("password must be at least 6 characters",{ variant: 'error' })

return false;

}

}

if(data.password!==data.confirmPassword){

enqueueSnackbar("Passwords do not match",{ variant: 'error' })

return false;

}

return true;

};

function handleFormData(e){

console.log(formData)

setFormData({...formData,[e.target.name] : e.target.value});

};

return (

<Box

display="flex"

flexDirection="column"

justifyContent="space-between"

minHeight="100vh"

>

<Header hasHiddenAuthButtons />

<Box className="content" minHeight="80vh">

<Stack spacing={2} className="form">

<h2 className="title">Register</h2>

<TextField

id="username"

label="Username"

variant="outlined"

title="Username"

name="username"

placeholder="Enter Username"

fullWidth

value={formData.username}

onChange={(e)=>{

handleFormData(e)

}}

/>

<TextField

id="password"

variant="outlined"

label="Password"

name="password"

type="password"

helperText="Password must be atleast 6 characters length"

fullWidth

placeholder="Enter a password with minimum 6 characters"

value={formData.password}

onChange={(e)=>{

handleFormData(e)

}}

/>

<TextField

id="confirmPassword"

variant="outlined"

label="Confirm Password"

name="confirmPassword"

type="password"

fullWidth

value={formData.confirmPassword}

onChange={(e)=>{

handleFormData(e)

}}

/>{isLoading?<Stack justifyContent={'center'} alignItems={"center"}><CircularProgress /></Stack>: <Button className="button" fullWidth variant="contained" onClick={()=> register(formData)}>

Register Now

</Button>}

<p className="secondary-action">

Already have an account?{" "}

<Link className="link" to="/login">

Login here

</Link>

</p>

</Stack>

</Box>

<Footer />

</Box>

);

};

export default Register;

REDISTER.CSS

/\* TODO: CRIO\_TASK\_MODULE\_REGISTER - For screens of min width 768px, display the form content to the right end of the screen \*/

.content {

background-image: url("../assets/hero\_image.png");

background-position: center;

background-repeat: no-repeat;

background-size: cover;

display: flex;

justify-content: center;

align-items: center;

height: 70vh;

}

.form {

background-color: white;

border-radius: 10px;

padding: 2rem;

min-width: 80%;

margin: 2rem;

box-shadow: 0px 0px 10px 0px rgba(0, 0, 0, 0.2);

}

.title {

color: #00a278;

}

.secondary-action {

padding: 1rem 0;

}

.link {

font-weight: 700;

text-decoration: none;

color: #45c09f;

}

.link:focus {

color: #00a278;

}

.link:active {

color: #45c09f;

}

.link:visited {

color: #45c09f;

}

.loading{

align-items: center;

}

@media screen and (min-width: 768px) {

.form {

min-width: 24rem;

}

.content {

justify-content: end;

}

}

PRODUCTS.JS

import { Search, SentimentDissatisfied } from "@mui/icons-material";

import ShoppingCartOutlinedIcon from "@mui/icons-material/ShoppingCartOutlined";

import { Stack } from "@mui/material";

import {

CircularProgress,

Grid,

InputAdornment,

TextField,

Paper,

} from "@mui/material";

import { Box } from "@mui/system";

import axios from "axios";

import { useSnackbar } from "notistack";

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

import { config } from "../App";

import Footer from "./Footer";

import Header from "./Header";

import ProductCard from "./ProductCard";

import "./Products.css";

import Cart, {generateCartItemsFrom}from "./Cart.js"

// Definition of Data Structures used

/\*\*

\* @typedef {Object} Product - Data on product available to buy

\*

\* @property {string} name - The name or title of the product

import "./Products.css";

/\*\*

\* @typedef {Object} CartItem - - Data on product added to cart

\*

\* @property {string} name - The name or title of the product in cart

\* @property {string} qty - The quantity of product added to cart

\* @property {string} category - The category that the product belongs to

\* @property {number} cost - The price to buy the product

\* @property {number} rating - The aggregate rating of the product (integer out of five)

\* @property {string} image - Contains URL for the product image

\* @property {string} \_id - Unique ID for the product

\*/

const Products = () => {

const { enqueueSnackbar } = useSnackbar();

const [product, setProduct] = useState([]);

const [isLoading, setIsLoading] = useState(false);

const [userSearch, setUserSearch] = useState("");

const [debounceTimer, setDebounceTimer] = useState(0);

const [cartData,setCartData]=useState([]);

const [productItem, setProductItem] = useState([]);

useEffect(() => {

performAPICall();

fetchCart(localStorage.getItem("token"));

}, []);

// TODO: CRIO\_TASK\_MODULE\_PRODUCTS - Fetch products data and store it

/\*\*

\* Make API call to get the products list and store it to display the products

\*

\* @returns { Array.<Product> }

\* Array of objects with complete data on all available products

\*

\* API endpoint - "GET /products"

\*

\* Example for successful response from backend:

\* HTTP 200

\* [

\* {

\* "name": "iPhone XR",

\* "category": "Phones",

\* "cost": 100,

\* "rating": 4,

\* "image": "https://i.imgur.com/lulqWzW.jpg",

\* "\_id": "v4sLtEcMpzabRyfx"

\* },

\* {

\* "name": "Basketball",

\* "category": "Sports",

\* "cost": 100,

\* "rating": 5,

\* "image": "https://i.imgur.com/lulqWzW.jpg",

\* "\_id": "upLK9JbQ4rMhTwt4"

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 500

\* {

\* "success": false,

\* "message": "Something went wrong. Check the backend console for more details"

\* }

\*/

const performAPICall = async () => {

setIsLoading(true);

try {

let res = await axios.get(`${config.endpoint}/products`);

console.log(res);

setProductItem(res.data);

setProduct(res.data);

} catch (error) {

if (error.response) {

enqueueSnackbar(`${error.response.data.message}`, { variant: "error" });

} else

enqueueSnackbar(

"Something went wrong. Check that the backend is running, reachable and returns valid JSON.",

{ variant: "error" }

);

} finally {

setIsLoading(false);

}

};

// TODO: CRIO\_TASK\_MODULE\_PRODUCTS - Implement search logic

/\*\*

\* Definition for search handler

\* This is the function that is called on adding new search keys

\*

\* @param {string} text

\* Text user types in the search bar. To filter the displayed products based on this text.

\*

\* @returns { Array.<Product> }

\* Array of objects with complete data on filtered set of products

\*

\* API endpoint - "GET /products/search?value=<search-query>"

\*

\*/

const performSearch = async (text) => {

setIsLoading(true);

try {

let res = await axios.get(

`${config.endpoint}/products/search?value=${text}`

);

console.log(res);

setProduct(res.data);

} catch (error) {

if (error.response) {

if (error.response.status === 404) {

setProduct([]);

} else {

enqueueSnackbar(`${error.response.data.message}`, {

variant: "error",

});

}

} else

enqueueSnackbar(

"Something went wrong. Check that the backend is running, reachable and returns valid JSON.",

{ variant: "error" }

);

} finally {

setIsLoading(false);

}

};

// TODO: CRIO\_TASK\_MODULE\_PRODUCTS - Optimise API calls with debounce search implementation

/\*\*

\* Definition for debounce handler

\* With debounce, this is the function to be called whenever the user types text in the searchbar field

\*

\* @param {{ target: { value: string } }} event

\* JS event object emitted from the search input field

\*

\* @param {NodeJS.Timeout} debounceTimeout

\* Timer id set for the previous debounce call

\*

\*/

const debounceSearch = (event, debounceTimeout) => {

clearInterval(debounceTimeout);

const newTimer = setTimeout(() => {

performSearch(event.target.value);

}, 500);

setDebounceTimer(newTimer);

};

const inputhandler = (event) => {

setUserSearch(event.target.value);

debounceSearch(event, debounceTimer);

};

/\*\*

\* Perform the API call to fetch the user's cart and return the response

\*

\* @param {string} token - Authentication token returned on login

\*

\* @returns { Array.<{ productId: string, qty: number }> | null }

\* The response JSON object

\*

\* Example for successful response from backend:

\* HTTP 200

\* [

\* {

\* "productId": "KCRwjF7lN97HnEaY",

\* "qty": 3

\* },

\* {

\* "productId": "BW0jAAeDJmlZCF8i",

\* "qty": 1

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 401

\* {

\* "success": false,

\* "message": "Protected route, Oauth2 Bearer token not found"

\* }

\*/

const fetchCart = async (token) => {

if (!token) return;

try {

// TODO: CRIO\_TASK\_MODULE\_CART - Pass Bearer token inside "Authorization" header to get data from "GET /cart" API and return the response data

let response= await axios.get(`${config.endpoint}/cart`,{ 'headers': { 'Authorization': 'Bearer '+token } })

console.log(response);

setCartData(response.data);

} catch (e) {

if (e.response && e.response.status === 400) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

} else {

enqueueSnackbar(

"Could not fetch cart details. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

return null;

}

};

// TODO: CRIO\_TASK\_MODULE\_CART - Return if a product already exists in the cart

/\*\*

\* Return if a product already is present in the cart

\*

\* @param { Array.<{ productId: String, quantity: Number }> } items

\* Array of objects with productId and quantity of products in cart

\* @param { String } productId

\* Id of a product to be checked

\*

\* @returns { Boolean }

\* Whether a product of given "productId" exists in the "items" array

\*

\*/

const handleQuantity=(event)=>{

const productId=event.currentTarget.value;

const token=localStorage.getItem("token")

const name=event.currentTarget.name;

if (!token){

enqueueSnackbar(

"Login to add an item to the Cart.",

{

variant: "error",

}

);

}else if(isItemInCart(cartData,productId)){

if(name){

let quantity= cartData.find((id) => id.productId === productId).qty;

if (name==="add"){

quantity+=1;

}else{

quantity-=1;

}

console.log(quantity)

addToCart(token,cartData,product,productId,quantity)

}else{

enqueueSnackbar(

"Item already in cart. Use the cart sidebar to update quantity or remove item.",

{

variant: "error",

}

);

}

}

else{

addToCart(token,cartData,product,productId,0,{preventDuplicate:true})

}

}

const isItemInCart = (items, productId) => {

const result=items.some((each)=>{

return each.productId===productId

})

return result;

};

/\*\*

\* Perform the API call to add or update items in the user's cart and update local cart data to display the latest cart

\*

\* @param {string} token

\* Authentication token returned on login

\* @param { Array.<{ productId: String, quantity: Number }> } items

\* Array of objects with productId and quantity of products in cart

\* @param { Array.<Product> } products

\* Array of objects with complete data on all available products

\* @param {string} productId

\* ID of the product that is to be added or updated in cart

\* @param {number} qty

\* How many of the product should be in the cart

\* @param {boolean} options

\* If this function was triggered from the product card's "Add to Cart" button

\*

\* Example for successful response from backend:

\* HTTP 200 - Updated list of cart items

\* [

\* {

\* "productId": "KCRwjF7lN97HnEaY",

\* "qty": 3

\* },

\* {

\* "productId": "BW0jAAeDJmlZCF8i",

\* "qty": 1

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 404 - On invalid productId

\* {

\* "success": false,

\* "message": "Product doesn't exist"

\* }

\*/

const addToCart = async (

token,

items,

products,

productId,

qty,

options = { preventDuplicate: false }

) => {

if (options.preventDuplicate){

const updateData={productId:productId,qty:1};

let responseData=await performCartApi(token,updateData);

console.log(responseData)

const nonZeroCart=responseData.filter(each=>(each.qty!==null));

console.log(nonZeroCart)

setCartData(nonZeroCart);

}else{

const updateData={productId:productId,qty:qty};

let responseData=await performCartApi(token,updateData);

console.log(responseData)

const nonZeroCart=responseData.filter(each=>(each.qty!==null));

console.log(nonZeroCart)

setCartData(nonZeroCart);

}

};

const performCartApi= async (token,updateData)=>{

try {

console.log(updateData);

// TODO: CRIO\_TASK\_MODULE\_CART - Pass Bearer token inside "Authorization" header to get data from "GET /cart" API and return the response data

let response= await axios.post(`${config.endpoint}/cart`,updateData,{ 'headers': { 'Authorization': 'Bearer '+token } })

return response.data;

} catch (e) {

if (e.response && e.response.status === 400) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

} else {

enqueueSnackbar(

"Could not fetch cart details. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

return null;

}

}

console.log(productItem);

// let productPage = (

// <Grid container spacing={2} mt={1}>

// {product.map((each) => {

// return (

// <Grid item xs={6} md={4} lg={3}>

// <ProductCard product={each} key={each.name} />

// </Grid>

// );

// })}

// </Grid>

// );

// if (product.length === 0) {

// productPage = (

// <Stack className="loading">

// <SentimentDissatisfied />

// <p>No products found</p>

// </Stack>

// );

// }

// let loginCart=localStorage.getItem("token")?<Grid container direction="row">

// <Grid item className="product-grid" md={8} lg={9}>

// <Box className="hero">

// <p className="hero-heading">

// India’s <span className="hero-highlight">FASTEST DELIVERY</span>{" "}

// to your door step

// </p>

// </Box>

// <Box className="product-mid">

// {isLoading ? (

// <Stack className="loading">

// <CircularProgress />

// <p>Loading Products...</p>

// </Stack>

// ) : (

// productPage

// )}

// </Box>

// </Grid>

// <Grid item className="cart-con" md={4} lg={3}>

// <Paper elevation={3}>

// <Box className="cart-card">

// <Stack className="loading">

// <ShoppingCartOutlinedIcon />

// <p>Cart is empty.Add more items to the cart to checkout </p>

// </Stack>

// </Box>

// </Paper>

// </Grid>

// </Grid>:<Grid container direction="row">

// <Grid item className="product-grid" >

// <Box className="hero">

// <p className="hero-heading">

// India’s <span className="hero-highlight">FASTEST DELIVERY</span>{" "}

// to your door step

// </p>

// </Box>

// <Box className="product-mid">

// {isLoading ? (

// <Stack className="loading">

// <CircularProgress />

// <p>Loading Products...</p>

// </Stack>

// ) : (

// productPage

// )}

// </Box>

// </Grid>

// </Grid>

// if (!localStorage.getItem("token")){

// loginCart=<Grid container direction="row">

// <Grid item className="product-grid" md={9}>

// <Box className="hero">

// <p className="hero-heading">

// India’s <span className="hero-highlight">FASTEST DELIVERY</span>{" "}

// to your door step

// </p>

// </Box>

// <Box className="product-mid">

// {isLoading ? (

// <Stack className="loading">

// <CircularProgress />

// <p>Loading Products...</p>

// </Stack>

// ) : (

// productPage

// )}

// </Box>

// </Grid>}

let productPage = (

<Grid container spacing={2} mt={1}>

{product.map((each) => {

return (

<Grid item xs={6} md={4} lg={3}>

<ProductCard product={each} key={each.name} handleAddToCart={handleQuantity} />

</Grid>

);

})}

</Grid>

);

if (product.length === 0) {

productPage = (

<Stack className="loading">

<SentimentDissatisfied />

<p>No products found</p>

</Stack>

);

}

let loginCart=localStorage.getItem("token")?

<Grid container direction="row">

<Grid item className="product-grid" md={8} lg={9}>

<Box className="hero">

<p className="hero-heading">

India’s <span className="hero-highlight">FASTEST DELIVERY</span>{" "}

to your door step

</p>

</Box>

<Box className="product-mid">

{isLoading ? (

<Stack className="loading">

<CircularProgress />

<p>Loading Products...</p>

</Stack>

) : (

productPage

)}

</Box>

</Grid>

<Grid item className="cart-con" md={4} lg={3}>

<Paper elevation={3} className="cart-bg">

<Box className="cart-card">

<Cart products={productItem} items={cartData} handleQuantity={handleQuantity}/>

</Box>

</Paper>

</Grid>

</Grid>:<Grid container direction="row">

<Grid item className="product-grid" >

<Box className="hero">

<p className="hero-heading">

India’s <span className="hero-highlight">FASTEST DELIVERY</span>{" "}

to your door step

</p>

</Box>

<Box className="product-mid">

{isLoading ? (

<Stack className="loading">

<CircularProgress />

<p>Loading Products...</p>

</Stack>

) : (

productPage

)}

</Box>

</Grid>

</Grid>

return (

<div>

<Header children={{ search: userSearch, fun: inputhandler }}></Header>

<TextField

className="search-mobile"

size="small"

fullWidth

InputProps={{

endAdornment: (

<InputAdornment position="end">

<Search color="primary" />

</InputAdornment>

),

}}

placeholder="Search for items/categories"

name="search"

value={userSearch}

onChange={(e) => inputhandler(e)}

/>

{loginCart}

<Footer />

</div>

);

};

export default Products;

PRODUCTS.CSS

.search {

min-width: 24rem;

position: relative;

left: 4rem;

}

.hero {

background-image: url("../assets/hero\_image.png");

background-position: center;

background-repeat: no-repeat;

background-size: cover;

height: 10rem;

display: flex;

justify-content: center;

align-items: center;

}

.cart-con{

width: 100%;

background-color: #E9F5E1;

padding: 8px;

}

.cart-bg{

position: sticky !important;

top: 0;

bottom: 0;

}

.cart-card{

min-height:35vh;

padding: 8px;

font-size: 1rem;

text-align: center;

}

.product-mid{

min-height:40vh;

}

.hero-heading {

color: #fff;

font-size: 1.5rem;

width: 80%;

position: relative;

left: 1rem;

}

.hero-highlight {

color: #212121;

font-weight: 700;

background: #ffb825;

border-radius: 5px;

padding: 0.2rem;

}

.loading {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

width: 100%;

height: 50vh;

}

.search-mobile {

display: block !important;

}

.search-desktop {

display: none !important;

}

@media screen and (min-width: 768px) {

.hero {

justify-content: flex-end;

height: 20rem;

padding-right: 8rem;

}

.hero-heading {

position: static;

font-size: 3rem;

width: 50%;

}

.search-mobile {

display: none !important;

}

.search-desktop {

display: block !important;

}

.product-grid {

}

.cart-con{

background-color: #E9F5E1;

padding: 8px;

}

.cart-card{

min-height:60vh;

padding: 8px;

font-size: 1rem;

}

}

PRODUCTCARD.JS

import { AddShoppingCartOutlined } from "@mui/icons-material";

import {

Button,

Card,

CardActions,

CardContent,

CardMedia,

Rating,

CardActionArea,

Typography,

} from "@mui/material";

import React from "react";

import "./ProductCard.css";

const ProductCard = ({ product, handleAddToCart }) => {

return (

<Card className="card" spacing={2}>

<CardMedia

component="img"

height="200"

image={product.image}

alt="Tan Leatherette Weekender Duffle"

/>

<CardContent>

<Typography gutterBottom variant="h5" component="div">

{product.name}

</Typography>

<Typography variant="body2" color="text.secondary">

<span>${product.cost}</span>

</Typography>

<Rating name="read-only" value={product.rating} readOnly />

</CardContent>

<CardActions>

<Button size="large" variant="contained" startIcon={<AddShoppingCartOutlined/>} fullWidth value={product.\_id} onClick={(e)=>handleAddToCart(e)}>ADD TO CART</Button>

</CardActions>

</Card>

);

};

export default ProductCard;

PRODUCTCARD.CSS

.card {

height: 100%;

display: flex;

flex-direction: column;

justify-content: space-between;

}

.card-actions {

margin-bottom: 1rem;

}

.card-button {

text-transform: none;

font-weight: 600;

}

CART.JS

import {

AddOutlined,

RemoveOutlined,

ShoppingCart,

ShoppingCartOutlined,

} from "@mui/icons-material";

import { Button, IconButton, Stack } from "@mui/material";

import { Box } from "@mui/system";

import React from "react";

import { useHistory } from "react-router-dom";

import "./Cart.css";

// Definition of Data Structures used

/\*\*

\* @typedef {Object} Product - Data on product available to buy

\*

\* @property {string} name - The name or title of the product

\* @property {string} category - The category that the product belongs to

\* @property {number} cost - The price to buy the product

\* @property {number} rating - The aggregate rating of the product (integer out of five)

\* @property {string} image - Contains URL for the product image

\* @property {string} \_id - Unique ID for the product

\*/

/\*\*

\* @typedef {Object} CartItem - - Data on product added to cart

\*

\* @property {string} name - The name or title of the product in cart

\* @property {string} qty - The quantity of product added to cart

\* @property {string} category - The category that the product belongs to

\* @property {number} cost - The price to buy the product

\* @property {number} rating - The aggregate rating of the product (integer out of five)

\* @property {string} image - Contains URL for the product image

\* @property {string} productId - Unique ID for the product

\*/

/\*\*

\* Returns the complete data on all products in cartData by searching in productsData

\*

\* @param { Array.<{ productId: String, qty: Number }> } cartData

\* Array of objects with productId and quantity of products in cart

\*

\* @param { Array.<Product> } productsData

\* Array of objects with complete data on all available products

\*

\* @returns { Array.<CartItem> }

\* Array of objects with complete data on products in cart

\*

\*/

export const generateCartItemsFrom = (cartData, productsData) => {

let cartItem = productsData.filter((each) => {

return cartData.some((id) => id.productId === each.\_id);

});

cartItem = cartItem.map((each) => {

return {

...each,

qty: cartData.find((id) => each.\_id === id.productId).qty,

productId: cartData.find((id) => each.\_id === id.productId).productId,

};

});

return cartItem;

};

/\*\*

\* Get the total value of all products added to the cart

\*

\* @param { Array.<CartItem> } items

\* Array of objects with complete data on products added to the cart

\*

\* @returns { Number }

\* Value of all items in the cart

\*

\*/

export const getTotalCartValue = (items = []) => {

const total = items.reduce((total, next) =>

total + (next.cost\*next.qty)

,0);

console.log(total);

return total;

};

/\*\*

\* Component to display the current quantity for a product and + and - buttons to update product quantity on cart

\*

\* @param {Number} value

\* Current quantity of product in cart

\*

\* @param {Function} handleAdd

\* Handler function which adds 1 more of a product to cart

\*

\* @param {Function} handleDelete

\* Handler function which reduces the quantity of a product in cart by 1

\*

\*

\*/

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Implement function to return total cart quantity

/\*\*

\* Return the sum of quantities of all products added to the cart

\*

\* @param { Array.<CartItem> } items

\* Array of objects with complete data on products in cart

\*

\* @returns { Number }

\* Total quantity of products added to the cart

\*

\*/

export const getTotalItems = (items = []) => {

};

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Add static quantity view for Checkout page cart

/\*\*

\* Component to display the current quantity for a product and + and - buttons to update product quantity on cart

\*

\* @param {Number} value

\* Current quantity of product in cart

\*

\* @param {Function} handleAdd

\* Handler function which adds 1 more of a product to cart

\*

\* @param {Function} handleDelete

\* Handler function which reduces the quantity of a product in cart by 1

\*

\* @param {Boolean} isReadOnly

\* If product quantity on cart is to be displayed as read only without the + - options to change quantity

\*

\*/

const ItemQuantity = ({ value, handleAdd, handleDelete,items }) => {

return (

<Stack direction="row" alignItems="center">

<IconButton size="small" color="primary" name="delete" value={items.productId} onClick={(e)=>handleDelete(e)} >

<RemoveOutlined />

</IconButton>

<Box padding="0.5rem" data-testid="item-qty">

{value}

</Box>

<IconButton size="small" color="primary" name="add" value={items.productId} onClick={(e)=>handleAdd(e)}>

<AddOutlined />

</IconButton>

</Stack>

);

};

/\*\*

\* Component to display the Cart view

\*

\* @param { Array.<Product> } products

\* Array of objects with complete data of all available products

\*

\* @param { Array.<Product> } items

\* Array of objects with complete data on products in cart

\*

\* @param {Function} handleDelete

\* Current quantity of product in cart

\*

\*

\*/

const CartProduct = ({ items, handleQty }) => {

return (<>

{handleQty?<Box display="flex" alignItems="flex-start" padding="1rem">

<Box className="image-container">

<img

// Add product image

src={items.image}

// Add product name as alt eext

alt={items.name}

width="100%"

height="100%"

/>

</Box>

<Box

display="flex"

flexDirection="column"

justifyContent="space-between"

height="6rem"

paddingX="1rem"

>

<div>{items.name}</div>

<Box display="flex" justifyContent="space-between" alignItems="center">

<ItemQuantity

value={items.qty}

handleAdd={handleQty}

handleDelete={handleQty}

items={items}

// Add required props by checking implementation

/>

<Box padding="0.5rem" fontWeight="700">

${items.cost}

</Box>

</Box>

</Box>

</Box>:<Box display="flex" alignItems="flex-start" padding="1rem">

<Box className="image-container">

<img

// Add product image

src={items.image}

// Add product name as alt eext

alt={items.name}

width="100%"

height="100%"

/>

</Box>

<Box

display="flex"

flexDirection="column"

justifyContent="space-between"

height="6rem"

paddingX="1rem"

>

<div>{items.name}</div>

<Box display="flex" justifyContent="space-between" alignItems="center">

<Box padding="0.5rem" fontWeight="700">

qty:{items.qty}

</Box>

<Box padding="0.5rem" fontWeight="700">

${items.cost}

</Box>

</Box>

</Box>

</Box>}

</>

);

};

const Cart = ({ products, items = [], handleQuantity }) => {

console.log(items);

const history = useHistory();

const cartItem = generateCartItemsFrom(items, products);

console.log(products);

if (!items.length) {

return (

<Box className="cart empty">

<ShoppingCartOutlined className="empty-cart-icon" />

<Box color="#aaa" textAlign="center">

Cart is empty. Add more items to the cart to checkout.

</Box>

</Box>

);

}

return (

<>

{handleQuantity?<Box className="cart">

{/\* TODO: CRIO\_TASK\_MODULE\_CART - Display view for each cart item with non-zero quantity \*/}

<Box>

{cartItem.map((each) => {

return <CartProduct items={each} key={each.name} handleQty={handleQuantity}/>;

})}

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Order total

</Box>

<Box

color="#3C3C3C"

fontWeight="700"

fontSize="1.5rem"

alignSelf="center"

data-testid="cart-total"

>

${getTotalCartValue(cartItem)}

</Box>

</Box>

<Box display="flex" justifyContent="flex-end" className="cart-footer">

<Button

color="primary"

variant="contained"

startIcon={<ShoppingCart />}

className="checkout-btn"

onClick={() => history.push("/checkout")}

>

Checkout

</Button>

</Box>

</Box>:

<Box>

<Box className="cart">

<Box >

{cartItem.map((each) => {

return <CartProduct items={each} key={each.name} />;

})}

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Order total

</Box>

<Box

color="#3C3C3C"

fontWeight="700"

fontSize="1.5rem"

alignSelf="center"

data-testid="cart-total"

>

${getTotalCartValue(cartItem)}

</Box>

</Box>

</Box>

<Box className="cart">

<Box color="#3C3C3C" alignSelf="center" font-size="24px" padding="1rem">

Order Details

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Product

</Box>

<Box color="#3C3C3C" alignSelf="center">

{items.length}

</Box>

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Sub Total

</Box>

<Box

color="#3C3C3C" alignSelf="center"

>

${getTotalCartValue(cartItem)}

</Box>

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Shipping

</Box>

<Box

>

Free $0

</Box>

</Box>

<Box

padding="1rem"

display="flex"

justifyContent="space-between"

alignItems="center"

>

<Box color="#3C3C3C" alignSelf="center">

Total

</Box>

<Box

color="#3C3C3C"

fontWeight="700"

fontSize="1.5rem"

alignSelf="center"

data-testid="cart-total"

>

${getTotalCartValue(cartItem)}

</Box>

</Box>

</Box>

</Box>}

</>

);

};

export default Cart;

CART.CSS

.cart {

background-color: #ffffff;

border-radius: 4px;

margin: 0.5rem;

}

.empty {

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

min-height: 60vh;

}

.cart-footer {

padding: 1rem;

}

.image-container {

max-height: 10rem;

max-width: 8rem;

}

.checkout-btn {

text-transform: none;

font-weight: 700;

font-size: 1.1rem;

}

.empty-cart-icon {

width: 3rem !important;

height: 3rem !important;

color: #aaa;

}

.cart-row {

display: flex;

align-items: center;

justify-content: space-between;

}

CHECKOUT.JS

import { CreditCard, Delete } from "@mui/icons-material";

import {

Button,

Divider,

Grid,

Stack,

TextField,

Typography,

} from "@mui/material";

import { Box } from "@mui/system";

import axios from "axios";

import { useSnackbar } from "notistack";

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

import { useHistory } from "react-router-dom";

import { config } from "../App";

import Cart, { getTotalCartValue, generateCartItemsFrom } from "./Cart";

import "./Checkout.css";

import Footer from "./Footer";

import Header from "./Header";

// Definition of Data Structures used

/\*\*

\* @typedef {Object} Product - Data on product available to buy

\*

\* @property {string} name - The name or title of the product

\* @property {string} category - The category that the product belongs to

\* @property {number} cost - The price to buy the product

\* @property {number} rating - The aggregate rating of the product (integer out of five)

\* @property {string} image - Contains URL for the product image

\* @property {string} \_id - Unique ID for the product

\*/

/\*\*

\* @typedef {Object} CartItem - - Data on product added to cart

\*

\* @property {string} name - The name or title of the product in cart

\* @property {string} qty - The quantity of product added to cart

\* @property {string} category - The category that the product belongs to

\* @property {number} cost - The price to buy the product

\* @property {number} rating - The aggregate rating of the product (integer out of five)

\* @property {string} image - Contains URL for the product image

\* @property {string} productId - Unique ID for the product

\*/

/\*\*

\* @typedef {Object} Address - Data on added address

\*

\* @property {string} \_id - Unique ID for the address

\* @property {string} address - Full address string

\*/

/\*\*

\* @typedef {Object} Addresses - Data on all added addresses

\*

\* @property {Array.<Address>} all - Data on all added addresses

\* @property {string} selected - Id of the currently selected address

\*/

/\*\*

\* @typedef {Object} NewAddress - Data on the new address being typed

\*

\* @property { Boolean } isAddingNewAddress - If a new address is being added

\* @property { String} value - Latest value of the address being typed

\*/

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Should allow to type a new address in the text field and add the new address or cancel adding new address

/\*\*

\* Returns the complete data on all products in cartData by searching in productsData

\*

\* @param { String } token

\* Login token

\*

\* @param { NewAddress } newAddress

\* Data on new address being added

\*

\* @param { Function } handleNewAddress

\* Handler function to set the new address field to the latest typed value

\*

\* @param { Function } addAddress

\* Handler function to make an API call to add the new address

\*

\* @returns { JSX.Element }

\* JSX for the Add new address view

\*

\*/

const AddNewAddressView = ({

token,

newAddress,

handleNewAddress,

addAddress,

}) => {

return (

<Box display="flex" flexDirection="column">

<TextField

multiline

minRows={4}

placeholder="Enter your complete address"

onChange={(e)=>handleNewAddress({

...newAddress,

value:e.target.value

})}

/>

<Stack direction="row" my="1rem">

<Button

variant="contained"

onClick={()=>addAddress(token,newAddress)}

>

Add

</Button>

<Button

variant="text"

onClick={()=>

handleNewAddress({

...newAddress,

isAddingNewAddress:false

})}

>

Cancel

</Button>

</Stack>

</Box>

);

};

const Checkout = () => {

const token = localStorage.getItem("token");

const history = useHistory();

const { enqueueSnackbar } = useSnackbar();

const [items, setItems] = useState([]);

const [products, setProducts] = useState([]);

const [addresses, setAddresses] = useState({ all: [], selected: "" });

const [newAddress, setNewAddress] = useState({

isAddingNewAddress: false,

value: "",

});

console.log(newAddress);

// Fetch the entire products list

const getProducts = async () => {

try {

const response = await axios.get(`${config.endpoint}/products`);

setProducts(response.data);

return response.data;

} catch (e) {

if (e.response && e.response.status === 500) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

return null;

} else {

enqueueSnackbar(

"Could not fetch products. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

}

};

// Fetch cart data

const fetchCart = async (token) => {

if (!token) return;

try {

const response = await axios.get(`${config.endpoint}/cart`, {

headers: {

Authorization: `Bearer ${token}`,

},

});

return response.data;

} catch {

enqueueSnackbar(

"Could not fetch cart details. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

return null;

}

};

/\*\*

\* Fetch list of addresses for a user

\*

\* API Endpoint - "GET /user/addresses"

\*

\* Example for successful response from backend:

\* HTTP 200

\* [

\* {

\* "\_id": "",

\* "address": "Test address\n12th street, Mumbai"

\* },

\* {

\* "\_id": "BW0jAAeDJmlZCF8i",

\* "address": "New address \nKolam lane, Chennai"

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 401

\* {

\* "success": false,

\* "message": "Protected route, Oauth2 Bearer token not found"

\* }

\*/

const getAddresses = async (token) => {

if (!token) return;

try {

const response = await axios.get(`${config.endpoint}/user/addresses`, {

headers: {

Authorization: `Bearer ${token}`,

},

});

console.log(response.data);

setAddresses({ ...addresses, all: response.data });

return response.data;

} catch {

enqueueSnackbar(

"Could not fetch addresses. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

return null;

}

};

/\*\*

\* Handler function to add a new address and display the latest list of addresses

\*

\* @param { String } token

\* Login token

\*

\* @param { NewAddress } newAddress

\* Data on new address being added

\*

\* @returns { Array.<Address> }

\* Latest list of addresses

\*

\* API Endpoint - "POST /user/addresses"

\*

\* Example for successful response from backend:

\* HTTP 200

\* [

\* {

\* "\_id": "",

\* "address": "Test address\n12th street, Mumbai"

\* },

\* {

\* "\_id": "BW0jAAeDJmlZCF8i",

\* "address": "New address \nKolam lane, Chennai"

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 401

\* {

\* "success": false,

\* "message": "Protected route, Oauth2 Bearer token not found"

\* }

\*/

const addAddress = async (token, newAddress) => {

try {

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Add new address to the backend and display the latest list of addresses

console.log(token);

console.log(newAddress.value)

const addressPost={

"address":newAddress.value

}

let response= await axios.post(`${config.endpoint}/user/addresses`,addressPost,{ 'headers': { 'Authorization': 'Bearer '+token } })

console.log(response.data)

setNewAddress({

isAddingNewAddress: false,

value: "",

});

setAddresses({ ...addresses, all: response.data });

} catch (e) {

console.log(e.message)

if (e.response) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

} else {

enqueueSnackbar(

"Could not add this address. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

}

};

/\*\*

\* Handler function to delete an address from the backend and display the latest list of addresses

\*

\* @param { String } token

\* Login token

\*

\* @param { String } addressId

\* Id value of the address to be deleted

\*

\* @returns { Array.<Address> }

\* Latest list of addresses

\*

\* API Endpoint - "DELETE /user/addresses/:addressId"

\*

\* Example for successful response from backend:

\* HTTP 200

\* [

\* {

\* "\_id": "",

\* "address": "Test address\n12th street, Mumbai"

\* },

\* {

\* "\_id": "BW0jAAeDJmlZCF8i",

\* "address": "New address \nKolam lane, Chennai"

\* }

\* ]

\*

\* Example for failed response from backend:

\* HTTP 401

\* {

\* "success": false,

\* "message": "Protected route, Oauth2 Bearer token not found"

\* }

\*/

const deleteAddress = async (token, addressId) => {

try {

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Delete selected address from the backend and display the latest list of addresses

let response= await axios.delete(`${config.endpoint}/user/addresses/${addressId}`,{ 'headers': { 'Authorization': 'Bearer '+token } })

setAddresses({ ...addresses, all: response.data });

} catch (e) {

if (e.response) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

} else {

enqueueSnackbar(

"Could not delete this address. Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

}

};

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Validate request for checkout

/\*\*

\* Return if the request validation passed. If it fails, display appropriate warning message.

\*

\* Validation checks - show warning message with given text if any of these validation fails

\*

#add-new-btn {

text-transform: none;

font-weight: 700;

margin: 1rem 0;

}

.address-item {

display: flex;

align-items: center;

justify-content: space-between;

padding: 1rem;

margin: 1rem 0;

border-radius: 5px;

cursor: pointer;

}

.shipping-container {

padding: 2rem 1rem;

}

.selected {

border: 1px solid #00A278;

background-color: #E9F5E1;

}

.not-selected {

border: 1px solid #ddd;

}

p {

color: #3C3C3C;

}

@media screen and (min-width: 768px) {

.shipping-container {

padding: 2rem 4rem;

}

} \* "You do not have enough balance in your wallet for this purchase"

\*

\* 2. No addresses added for user

\* "Please add a new address before proceeding."

\*

\* 3. No address selected for checkout

\* "Please select one shipping address to proceed."

\*

\* @param { Array.<CartItem> } items

\* Array of objects with complete data on products added to the cart

\*

\* @param { Addresses } addresses

\* Contains data on array of addresses and selected address id

\*

\* @returns { Boolean }

\* Whether validation passed or not

\*

\*/

const validateRequest = (items, addresses) => {

let cartValue=getTotalCartValue(items);

let balance=localStorage.getItem("balance");

if (balance-cartValue<0){

enqueueSnackbar(

"You do not have enough balance in your wallet for this purchase",

{

variant: "error",

}

);

return false;

}

if (addresses.all.length===0){

enqueueSnackbar(

"Please add a new address before proceeding.",

{

variant: "error",

}

);

return false;

}

if (addresses.selected===""){

enqueueSnackbar(

"Please select one shipping address to proceed.",

{

variant: "error",

}

);

return false;

}

return true;

};

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT

/\*\*

\* Handler function to perform checkout operation for items added to the cart for the selected address

\*

\* @param { String } token

\* Login token

\*

\* @param { Array.<CartItem } items

\* Array of objects with complete data on products added to the cart

\*

\* @param { Addresses } addresses

\* Contains data on array of addresses and selected address id

\*

\* @returns { Boolean }

\* If checkout operation was successful

\*

\* API endpoint - "POST /cart/checkout"

\*

\* Example for successful response from backend:

\* HTTP 200

\* {

\* "success": true

\* }

\*

\* Example for failed response from backend:

\* HTTP 400

\* {

\* "success": false,

\* "message": "Wallet balance not sufficient to place order"

\* }

\*

\*/

const performCheckout = async (token, items, addresses) => {

if (validateRequest(items,addresses)){

const addressId={"addressId":addresses.selected}

try {

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Delete selected address from the backend and display the latest list of addresses

let response= await axios.post(`${config.endpoint}/cart/checkout`,addressId,{ 'headers': { 'Authorization': 'Bearer '+token } })

let cartValue=getTotalCartValue(items);

let balance=localStorage.getItem("balance");

localStorage.setItem("balance",balance-cartValue);

console.log(response);

history.push("/thanks")

} catch (e) {

if (e.response) {

enqueueSnackbar(e.response.data.message, { variant: "error" });

} else {

enqueueSnackbar(

"Check that the backend is running, reachable and returns valid JSON.",

{

variant: "error",

}

);

}

}

}

};

// TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Fetch addressses if logged in, otherwise show info message and redirect to Products page

// Fetch products and cart data on page load

useEffect(() => {

const onLoadHandler = async () => {

const productsData = await getProducts();

const cartData = await fetchCart(token);

if (productsData && cartData) {

const cartDetails = await generateCartItemsFrom(cartData, productsData);

setItems(cartDetails);

}

};

onLoadHandler();

// eslint-disable-next-line react-hooks/exhaustive-deps

getAddresses(token);

}, []);

const AddressCon=({address,addId})=>{

console.log(addId);

let classSelect="not-selected address-item";

if(addresses.selected===addId){

classSelect="selected address-item";

}

return <Box id={addId} onClick={(e)=>{setAddresses({...addresses,selected:addId})}}>

<Box className={classSelect} >

<Typography color="#3C3C3C" my="1rem">

{address}

</Typography>

<Box>

<Button startIcon={<Delete />} onClick={()=>deleteAddress(token,addId)} >

Delete

</Button></Box>

</Box>

</Box>

}

console.log(addresses);

const IsLoggedIn=()=>{

enqueueSnackbar(

"You must be logged in to access checkout page",

{

variant: "error",

}

);

history.push("/login");

return null;

}

return (<>

{token?(<>

<Header />

<Grid container>

<Grid item xs={12} md={9}>

<Box className="shipping-container" minHeight="100vh">

<Typography color="#3C3C3C" variant="h4" my="1rem">

Shipping

</Typography>

<Typography color="#3C3C3C" my="1rem">

Manage all the shipping addresses you want. This way you won't

have to enter the shipping address manually with every order.

Select the address you want to get your order delivered.

</Typography>

<Divider />

<Box>

{/\* TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Display list of addresses and corresponding "Delete" buttons, if present, of which 1 can be selected \*/}

{addresses.all.length!==0?

<Box>{addresses.all.map((each)=>{

return <AddressCon address={each.address} addId={each.\_id} key={each.\_id}/>

})}</Box>: <Typography my="1rem">

No addresses found for this account. Please add one to proceed

</Typography>}

</Box>

{/\* TODO: CRIO\_TASK\_MODULE\_CHECKOUT - Dislay either "Add new address" button or the <AddNewAddressView> component to edit the currently selected address \*/}

{newAddress.isAddingNewAddress? <AddNewAddressView

token={token}

newAddress={newAddress}

handleNewAddress={setNewAddress}

addAddress={addAddress}

/>:<Button

color="primary"

variant="contained"

id="add-new-btn"

size="large"

onClick={() => {

setNewAddress((currNewAddress) => ({

...currNewAddress,

isAddingNewAddress: true,

}));

}}

>

Add new address

</Button>}

<Typography color="#3C3C3C" variant="h4" my="1rem">

Payment

</Typography>

<Typography color="#3C3C3C" my="1rem">

Payment Method

</Typography>

<Divider />

<Box my="1rem">

<Typography>Wallet</Typography>

<Typography>

Pay ${getTotalCartValue(items)} of available $

{localStorage.getItem("balance")}

</Typography>

</Box>

<Button

startIcon={<CreditCard />}

variant="contained"

onClick={()=>{performCheckout(token,items,addresses)}}

>

PLACE ORDER

</Button>

</Box>

</Grid>

<Grid item xs={12} md={3} bgcolor="#E9F5E1">

<Cart isReadOnly products={products} items={items} />

</Grid>

</Grid>

<Footer />

</>):<IsLoggedIn/>}

</>

);

};

export default Checkout;

CHECKOUT.CSS

#add-new-btn {

text-transform: none;

font-weight: 700;

margin: 1rem 0;

}

.address-item {

display: flex;

align-items: center;

justify-content: space-between;

padding: 1rem;

margin: 1rem 0;

border-radius: 5px;

cursor: pointer;

}

.shipping-container {

padding: 2rem 1rem;

}

.selected {

border: 1px solid #00A278;

background-color: #E9F5E1;

}

.not-selected {

border: 1px solid #ddd;

}

p {

color: #3C3C3C;

}

@media screen and (min-width: 768px) {

.shipping-container {

padding: 2rem 4rem;

}

}