import { Purchase } from './../../models/purchase';

import { OrderItem } from './../../models/order-item';

import { CartItem } from './../../models/cart-item';

import { Customer } from './../../models/customer';

import { Order } from './../../models/order';

import { CheckoutService } from './../../services/checkout.service';

import { ShopValidators } from './../../validators/shop-validators';

import { State } from './../../models/state';

import { Country } from './../../models/country';

import { ShopFormService } from './../../services/shop-form.service';

import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';

import { CartService } from 'src/app/services/cart.service';

import { Router } from '@angular/router';

@Component({

selector: 'app-checkout',

templateUrl: './checkout.component.html',

styleUrls: ['./checkout.component.css']

})

export class CheckoutComponent implements OnInit {

checkoutFormGroup: FormGroup;

totalPrice: number = 0;

totalQuantity: number = 0;

creditCardYears: number[] = [];

creditCardMonths: number[] = [];

countries: Country[] = [];

shippingAddressStates: State[] = [];

billingAddressStates: State[] = [];

constructor(private formBuilder: FormBuilder,

private shopFormService: ShopFormService,

private cartService: CartService,

private checkoutService: CheckoutService,

private router: Router) { }

ngOnInit() {

this.checkoutFormGroup = this.formBuilder.group({

customer: this.formBuilder.group({

firstName: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

lastName: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

email: new FormControl('', [Validators.required, Validators.pattern('^[a-z0-9.\_%+-]+@[a-z0-9.-]+\\.[a-z]{2,4}$')])

}),

shippingAddress: this.formBuilder.group({

country: new FormControl('', [Validators.required]),

street: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

city: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

state: new FormControl('', [Validators.required]),

zipCode: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces])

}),

billingAddress: this.formBuilder.group({

country: new FormControl('', [Validators.required]),

street: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

city: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

state: new FormControl('', [Validators.required]),

zipCode: new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces])

}),

creditCard: this.formBuilder.group({

cardType: new FormControl('', [Validators.required]),

nameOnCard:new FormControl('', [Validators.required, Validators.minLength(2), ShopValidators.notOnlyWhiteSpaces]),

cardNumber: new FormControl('', [Validators.required, Validators.pattern('[0-9]{16}')]),

securityCode: new FormControl('', [Validators.required, Validators.pattern('[0-9]{3}')]),

expirationMonth: [''],

expirationYear: [''],

})

});

this.getCreditCardYears();

this.getCreditCardMonths();

this.getAllCountries();

this.reviewCartTotals();

}

reviewCartTotals() {

this.cartService.totalPrice.subscribe(

data => this.totalPrice = data

);

this.cartService.totalQuantity.subscribe(

data => this.totalQuantity = data

);

}

// getters for customers

get firstName(){ return this.checkoutFormGroup.get('customer.firstName');}

get lastName(){ return this.checkoutFormGroup.get('customer.lastName');}

get email(){ return this.checkoutFormGroup.get('customer.email');}

// getters for shipping Address

get shippingAddressCountry(){ return this.checkoutFormGroup.get('shippingAddress.country');}

get shippingAddressState(){ return this.checkoutFormGroup.get('shippingAddress.state');}

get shippingAddressStreet(){ return this.checkoutFormGroup.get('shippingAddress.street');}

get shippingAddressZipCode(){ return this.checkoutFormGroup.get('shippingAddress.zipCode');}

get shippingAddressCity(){ return this.checkoutFormGroup.get('shippingAddress.city');}

// getters for billing Address

get billingAddressCountry(){ return this.checkoutFormGroup.get('billingAddress.country');}

get billingAddressState(){ return this.checkoutFormGroup.get('billingAddress.state');}

get billingAddressStreet(){ return this.checkoutFormGroup.get('billingAddress.street');}

get billingAddressZipCode(){ return this.checkoutFormGroup.get('billingAddress.zipCode');}

get billingAddressCity(){ return this.checkoutFormGroup.get('billingAddress.city');}

// CreditCard getters

get creditCardType(){ return this.checkoutFormGroup.get('creditCard.cardType');}

get creditCardNameOnCard(){ return this.checkoutFormGroup.get('creditCard.nameOnCard');}

get creditCardNumber(){ return this.checkoutFormGroup.get('creditCard.cardNumber');}

get creditCardSecurityCode(){ return this.checkoutFormGroup.get('creditCard.securityCode');}

onSubmit(){

if(this.checkoutFormGroup.invalid){

this.checkoutFormGroup.markAllAsTouched();

return;

}

// set up order

let order= new Order();

order.totalPrice = this.totalPrice;

order.totalQuantity = this.totalQuantity;

// get cart items

const cartItems = this.cartService.cartItems;

// create orderItems from cartItems

let orderItems: OrderItem[] = cartItems.map(tempCartItem => new OrderItem(tempCartItem));

// set up purchase

let purchase = new Purchase();

// populate purchase - customer

purchase.customer = this.checkoutFormGroup.controls['customer'].value;

// populate purchase - shippingAddress

purchase.shippingAddress = this.checkoutFormGroup.controls['shippingAddress'].value;

const shippingState: State = JSON.parse(JSON.stringify(purchase.shippingAddress.state));

const shippingCountry: Country = JSON.parse(JSON.stringify(purchase.shippingAddress.country));

purchase.shippingAddress.state = shippingState.name;

purchase.shippingAddress.country = shippingCountry.name;

// populate purchase - billingAddress

purchase.billingAddress = this.checkoutFormGroup.controls['billingAddress'].value;

const billingState: State = JSON.parse(JSON.stringify(purchase.billingAddress.state));

const billingCountry: Country = JSON.parse(JSON.stringify(purchase.billingAddress.country));

purchase.billingAddress.state = billingState.name;

purchase.billingAddress.country = billingCountry.name;

// populate purchase - order and orderItems

purchase.order = order;

purchase.orderItems = orderItems;

// call REST API via checkoutService

this.checkoutService.placeOrder(purchase).subscribe(

data =>{

alert(`your order has been recieved.\n order tracking number: ${data.orderTrackingNumber}`);

// reset checkout form

this.resetCart();

},

error=>{

alert(`there was a error: ${error.message}`);

}

)

}

resetCart() {

// reset cart data

this.cartService.cartItems = [];

this.cartService.totalPrice.next(0);

this.cartService.totalQuantity.next(0);

// reset the form

this.checkoutFormGroup.reset();

// navigate back to the products page

this.router.navigateByUrl('/products');

}

copyShippingAddressToBillingAddress(event){

if(event.target.checked){

this.checkoutFormGroup.controls.billingAddress

.setValue(this.checkoutFormGroup.controls.shippingAddress.value);

this.billingAddressStates = this.shippingAddressStates;

}else{

this.checkoutFormGroup.controls.billingAddress.reset();

this.billingAddressStates = [];

}

}

getCreditCardYears(){

this.shopFormService.getCreditCardYears().subscribe(

data=>{

this.creditCardYears = data;

});

}

getCreditCardMonths(){

let startMonth: number = new Date().getMonth() + 1;

this.shopFormService.getCreditCardMonths(startMonth).subscribe(

data=>{

this.creditCardMonths = data;

});

}

onChangeYear(){

let currentYear: number = new Date().getFullYear();

let selectedYear: number = this.checkoutFormGroup.get('creditCard').value.expirationYear;

let startMonth: number;

// if the current year equals the selected year, then start with the current month.

if(currentYear == selectedYear){

startMonth = new Date().getMonth() + 1;

}else{

startMonth = 1;

}

this.shopFormService.getCreditCardMonths(startMonth).subscribe(

data=>{

this.creditCardMonths = data;

});

}

getAllCountries(){

this.shopFormService.getCountries().subscribe(

data=>{

this.countries = data;

}

)

}

getStates(formGroupName: string){

const formgroup = this.checkoutFormGroup.get(formGroupName);

const countryCode = formgroup.value.country.code;

const countryName = formgroup.value.country.name;

this.shopFormService.getStates(countryCode).subscribe(

data =>{

if(formGroupName === 'shippingAddress'){

this.shippingAddressStates = data;

}else{

this.billingAddressStates = data;

}

formgroup.get('state').setValue(data[0]);

});

}

}