Ecommerce website

Bianca Grama

Lungoci Luca

- **Introduction**

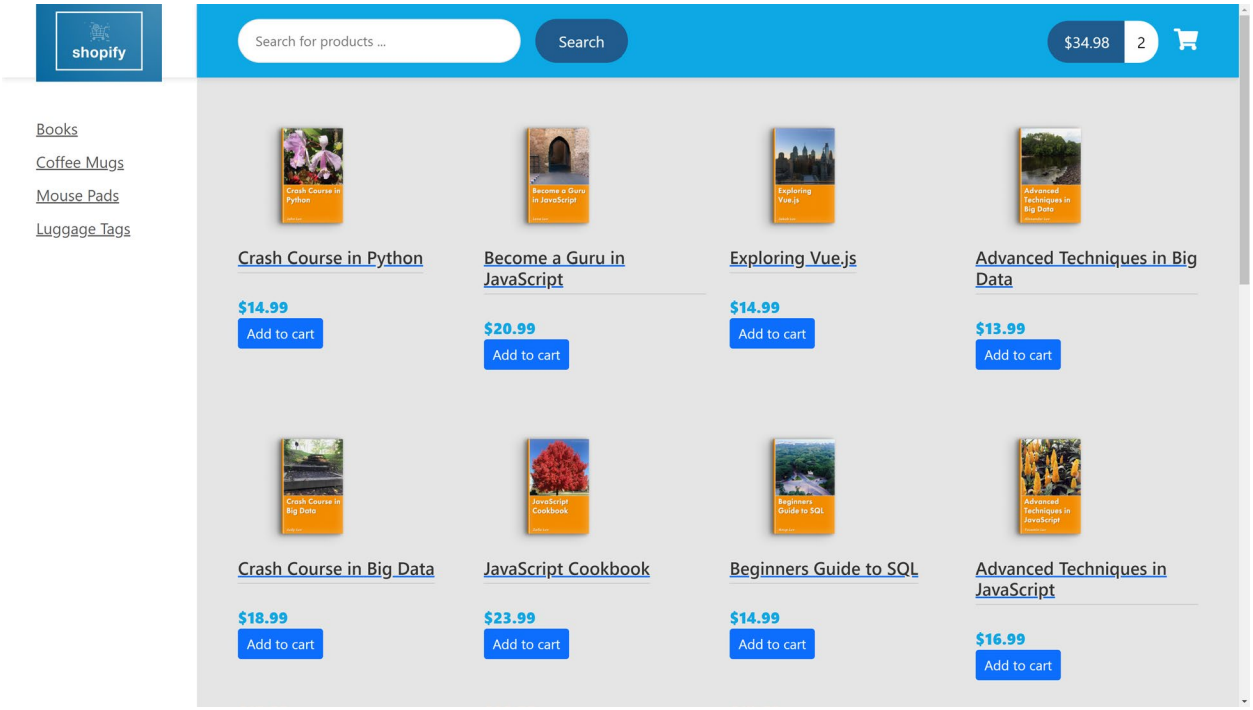
The aim of this project was to reproduce an ecommerce website such as “eBay” or “Amazon”. The basic functionalities of such a website are adding products to a cart, viewing products, performing an order. Another important aspect for our application was the user interface. Our website should be used by everybody, so it should be easy to use and pretty looking. From the moment the user enters our website, buttons to navigate through product categories, adding, viewing and searching products are available on the screen.

- **Short text**

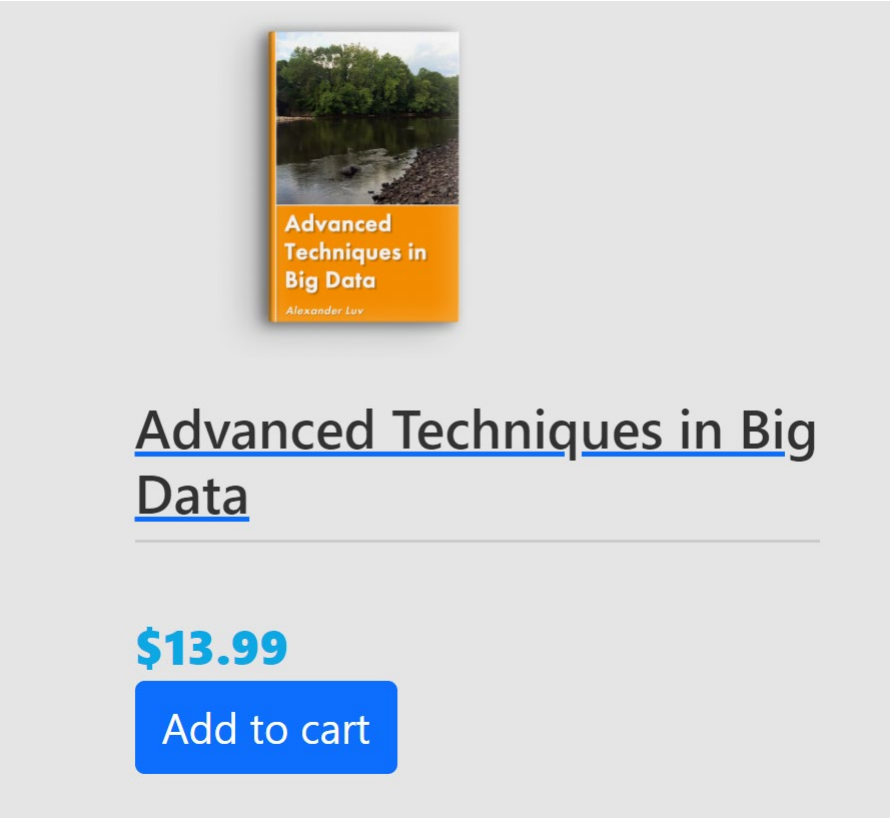
In the code section, the text highlighted with gray correspond to the classes written by my classmate.

I worked on the following: adding items to the cart (both from main page and detailed page), viewing items added in the cart, displaying and computing the number of items in the cart and total price, the checkout page and the functionalities in it, storing orders in the database and retrieving countries and states from the database.

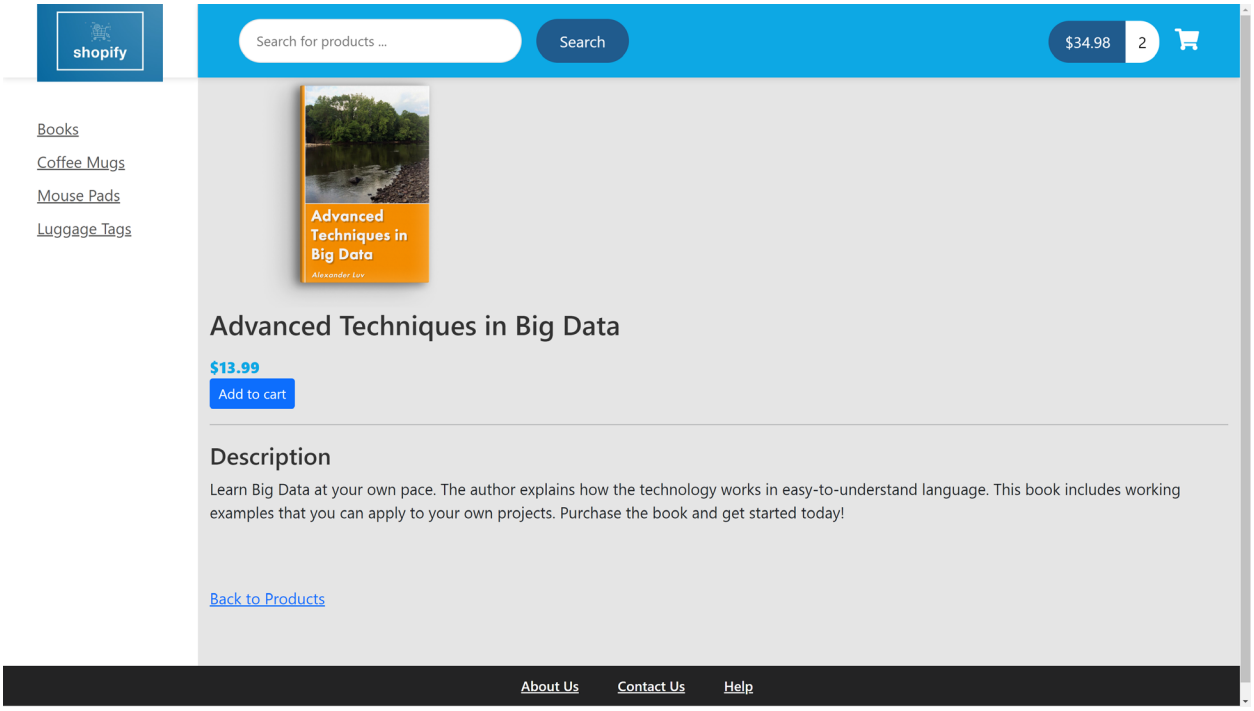
In the following section I will present in more details the functionalities of this application. In the main page there are loaded all the products form the selected category.



Shopping through our application is so easy, just click the “Add to cart” corresponding to the item you wish to purchase and it will be added to your order.

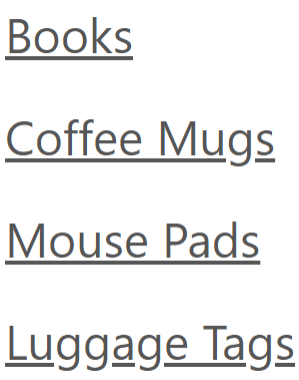


If you wish to take a closer look at our products, just click on them and you will be taken to a page when you can read more about it.

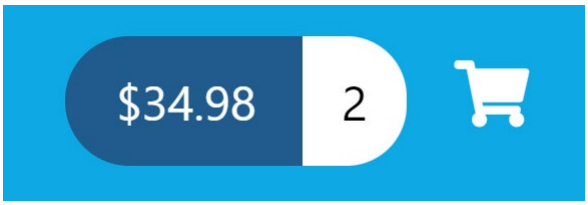


Users can search for certain products by entering their name and pressing “Search”.

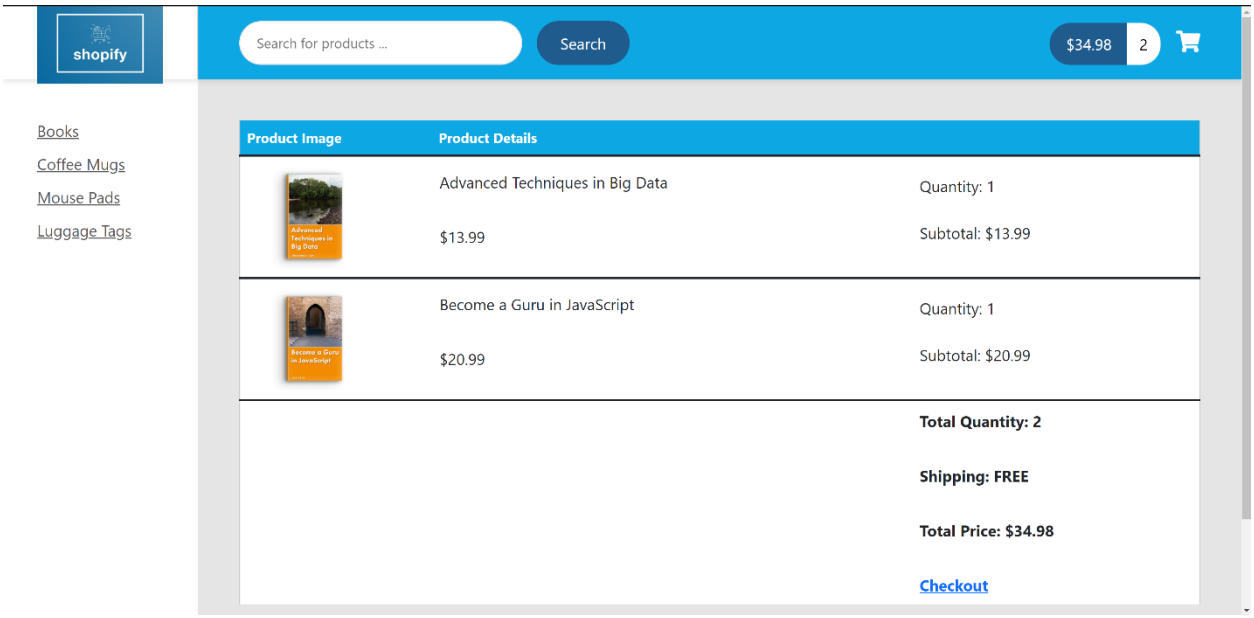
Users can switch between categories using this side menu.



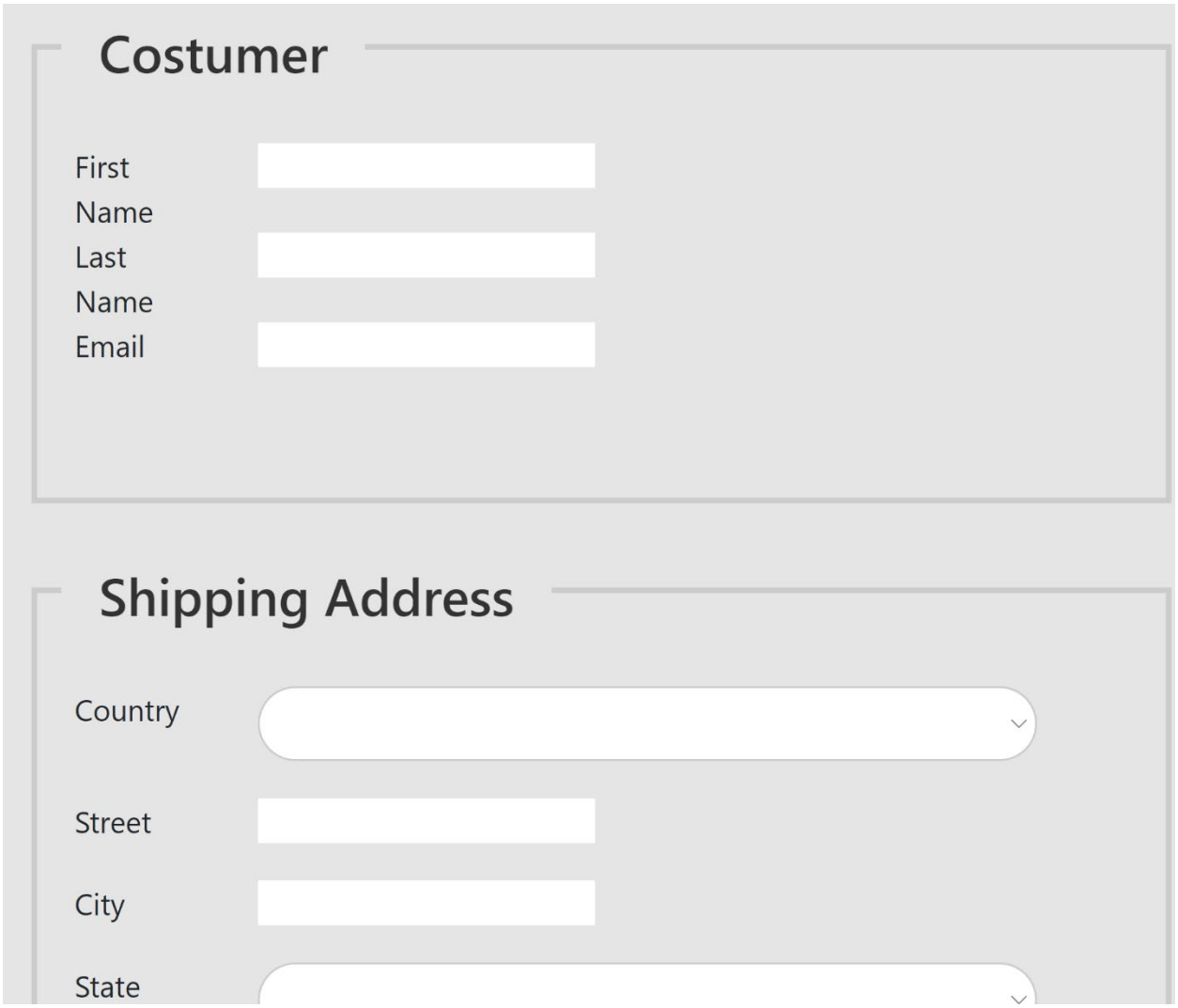
While shopping, you can see the current status of your shopping cart: the total price of your order and the total number added so far.



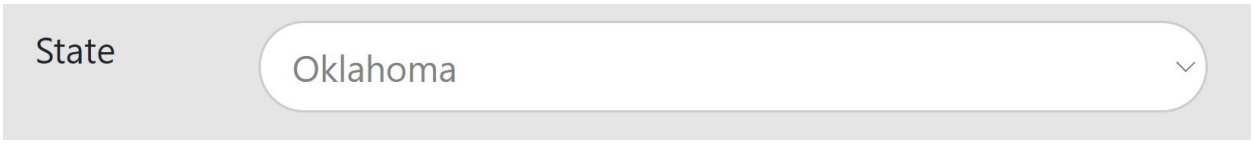
If you wish to take the next step and order the items, you can click the shopping cart icon and see your cart items.

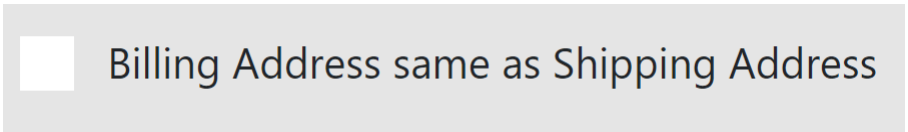


Hit the “Checkout” button and you will be presented with a checkout form.



Some additional functionalities include copying the shipping address data to billing address and drop-down lists for the country, state, credit card expiration date.





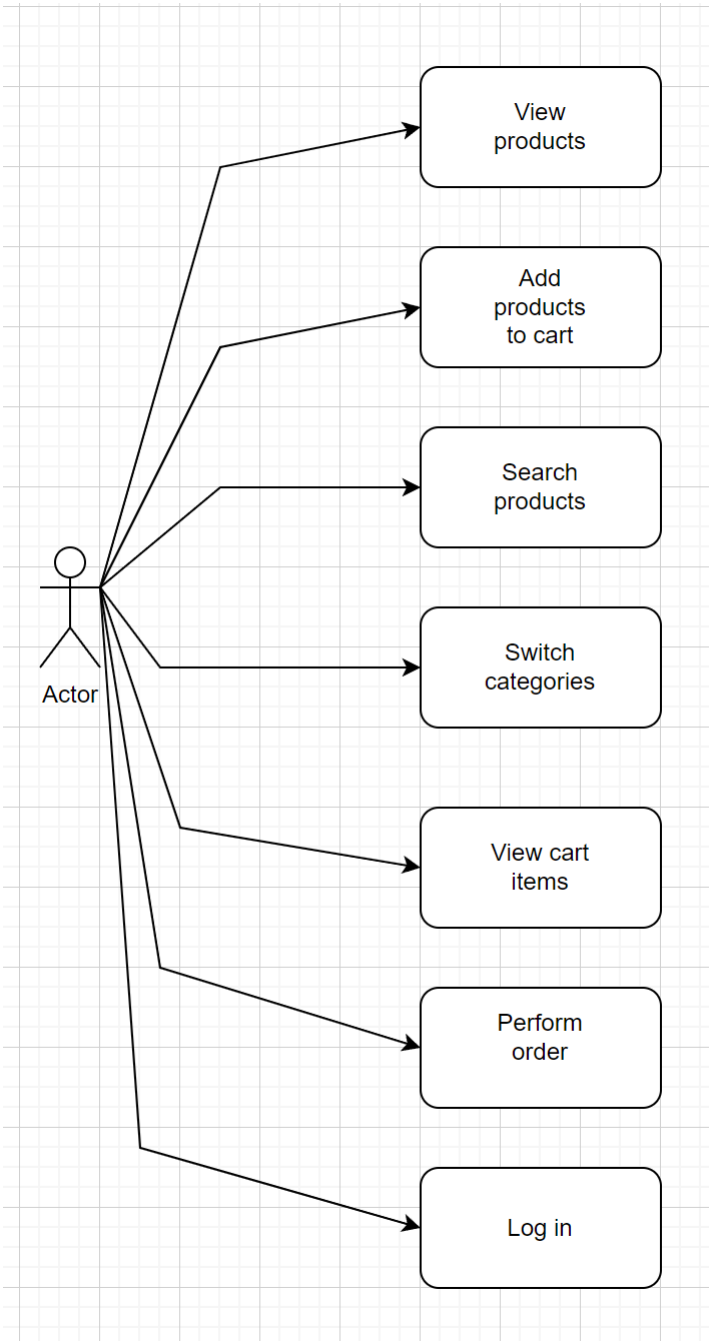
- **Bibliography**

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

https://www.tutorialspoint.com/spring_security/index.htm

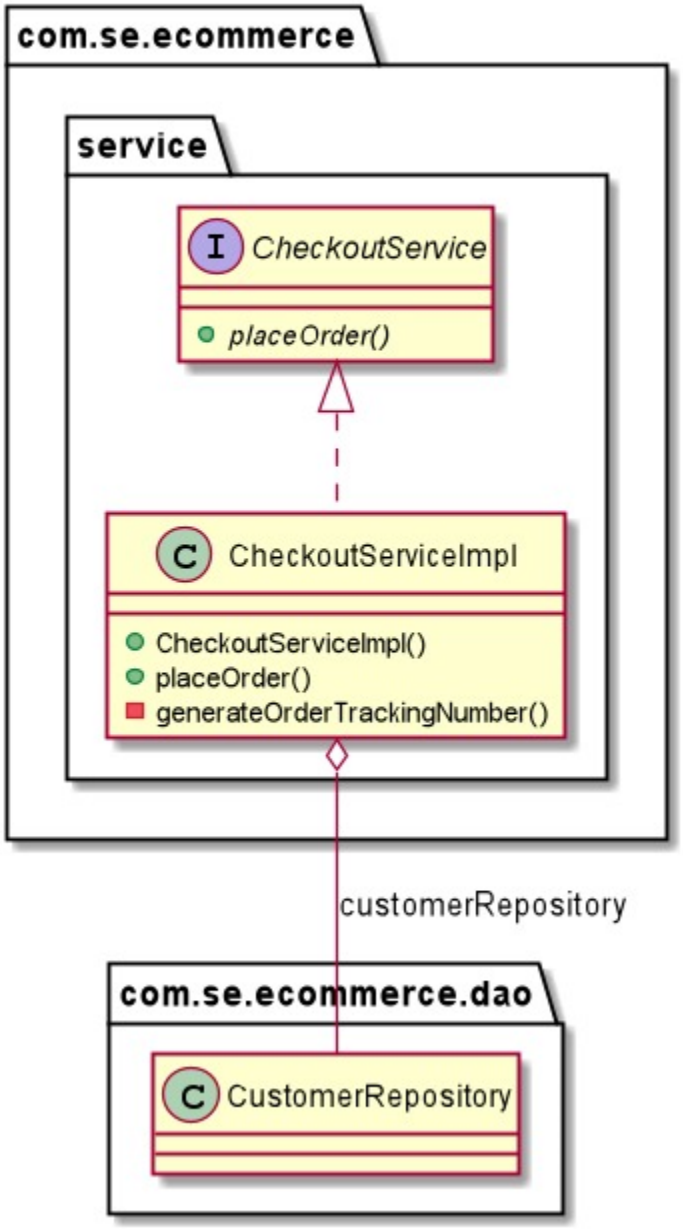
- **Diagrams**

Use case diagram

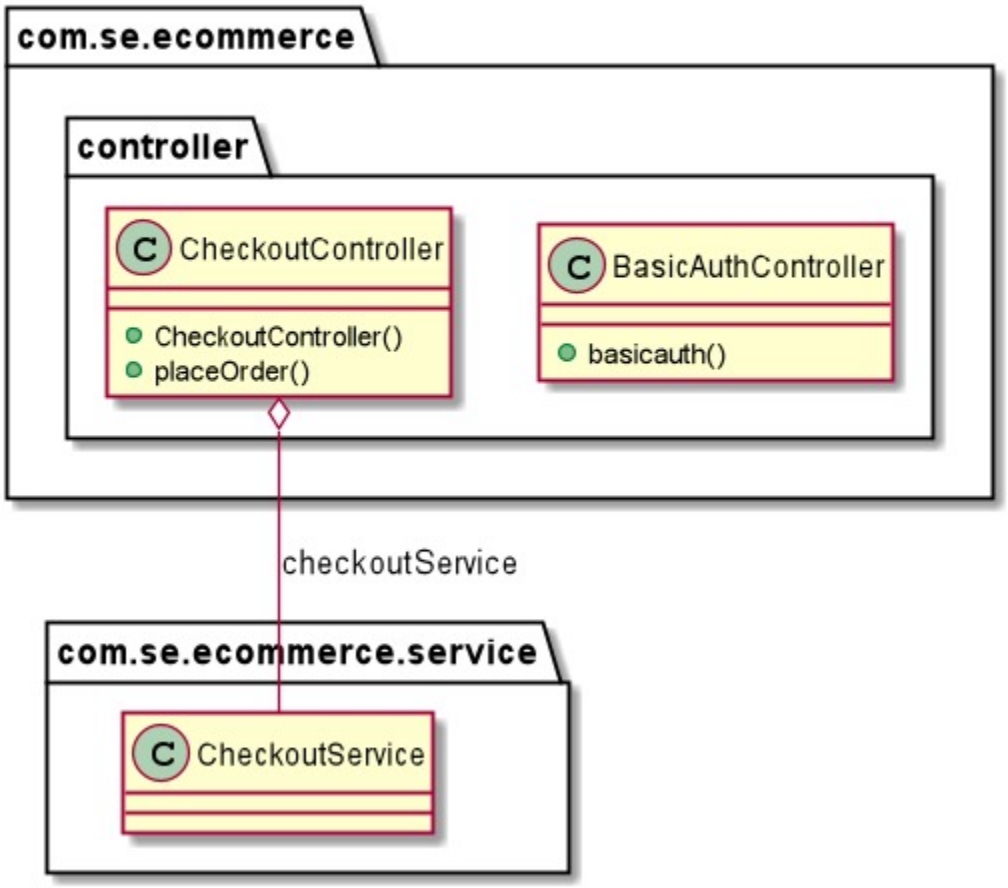


Class diagram

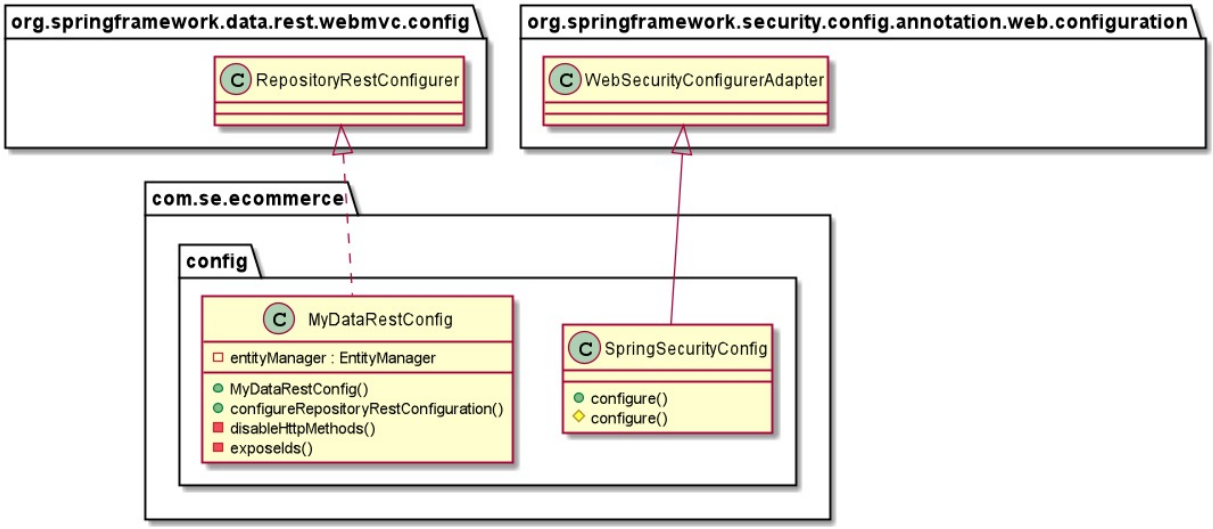
SERVICE's Class Diagram



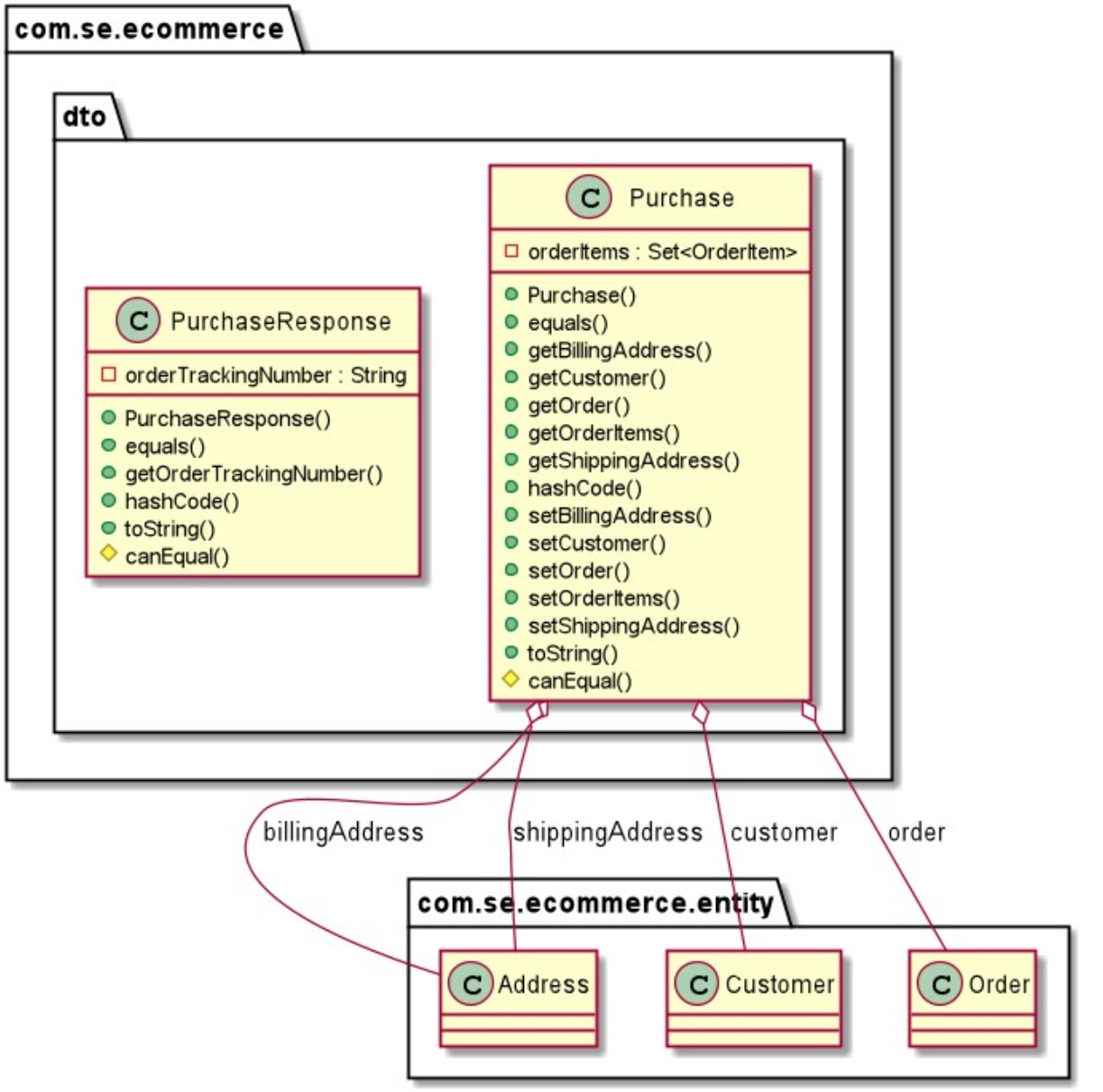
CONTROLLER's Class Diagram



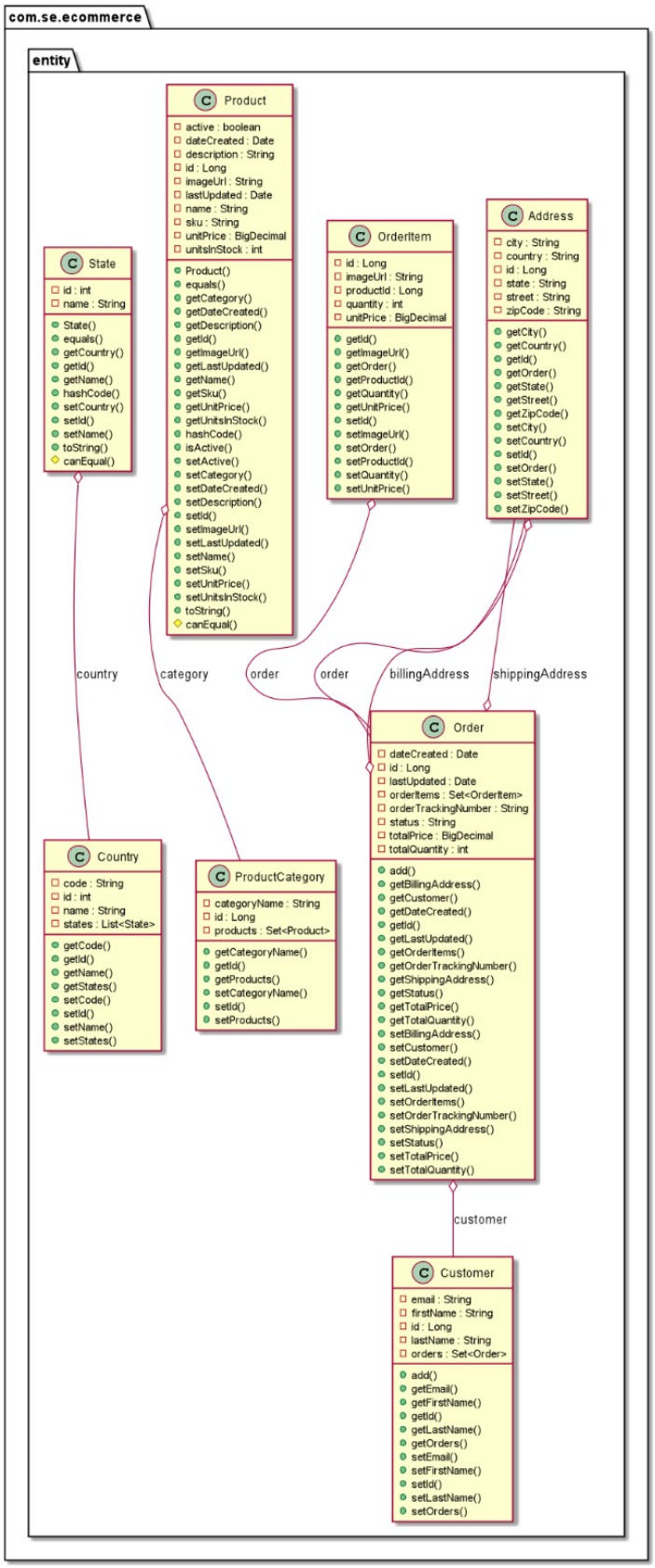
CONFIG's Class Diagram



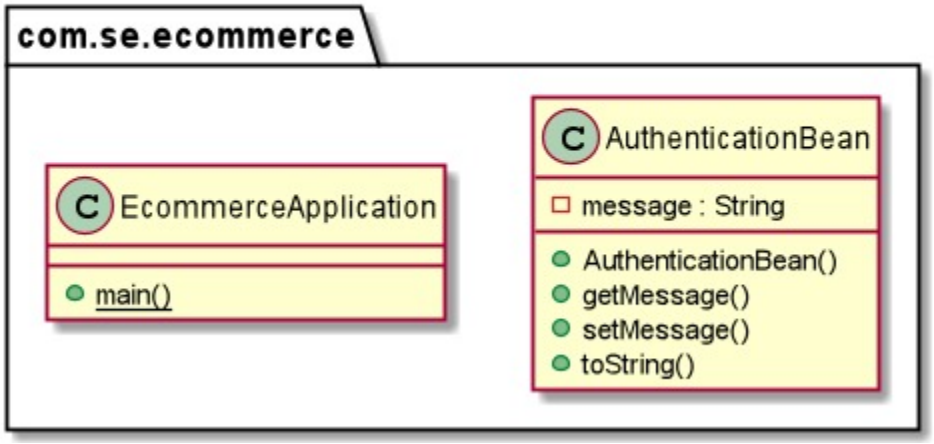
DTO's Class Diagram



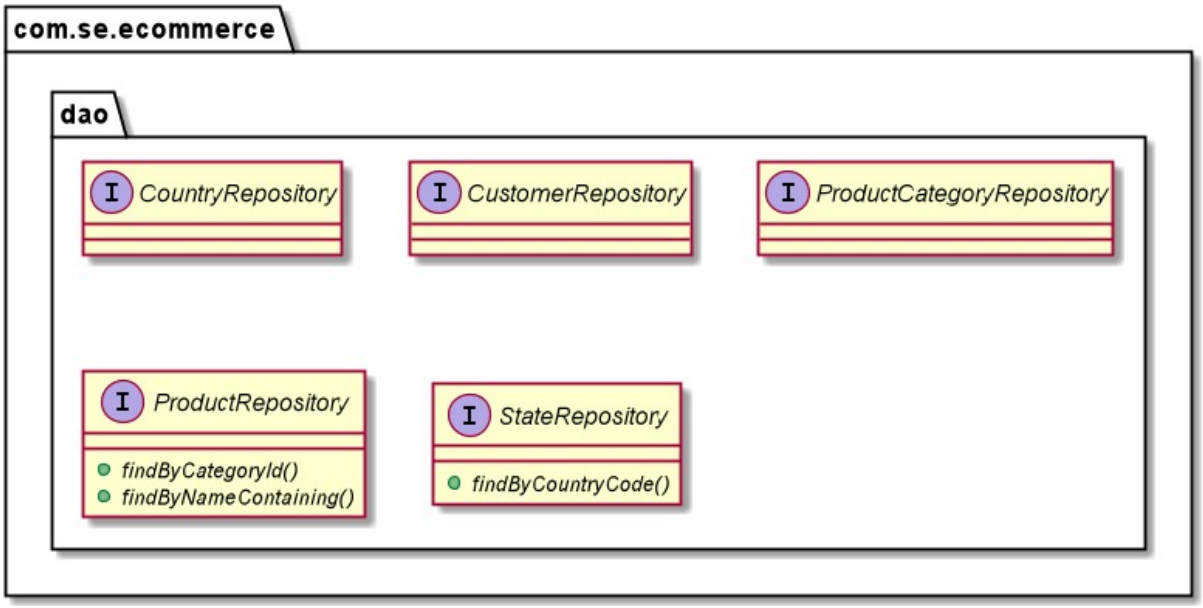
ENTITY's Class Diagram



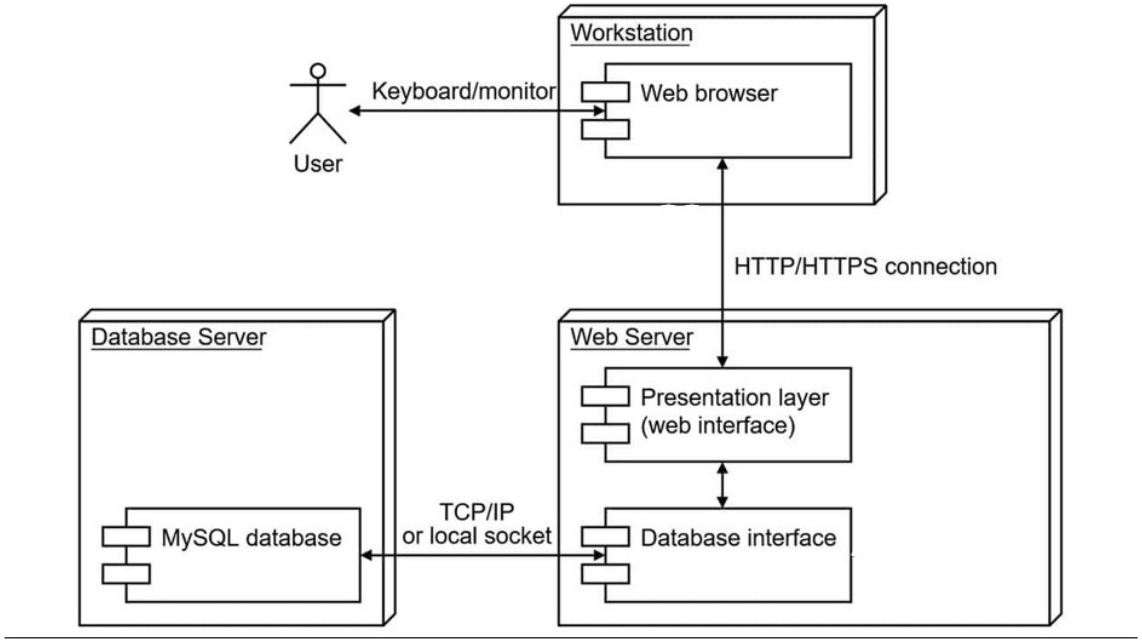
ECOMMERCE's Class Diagram



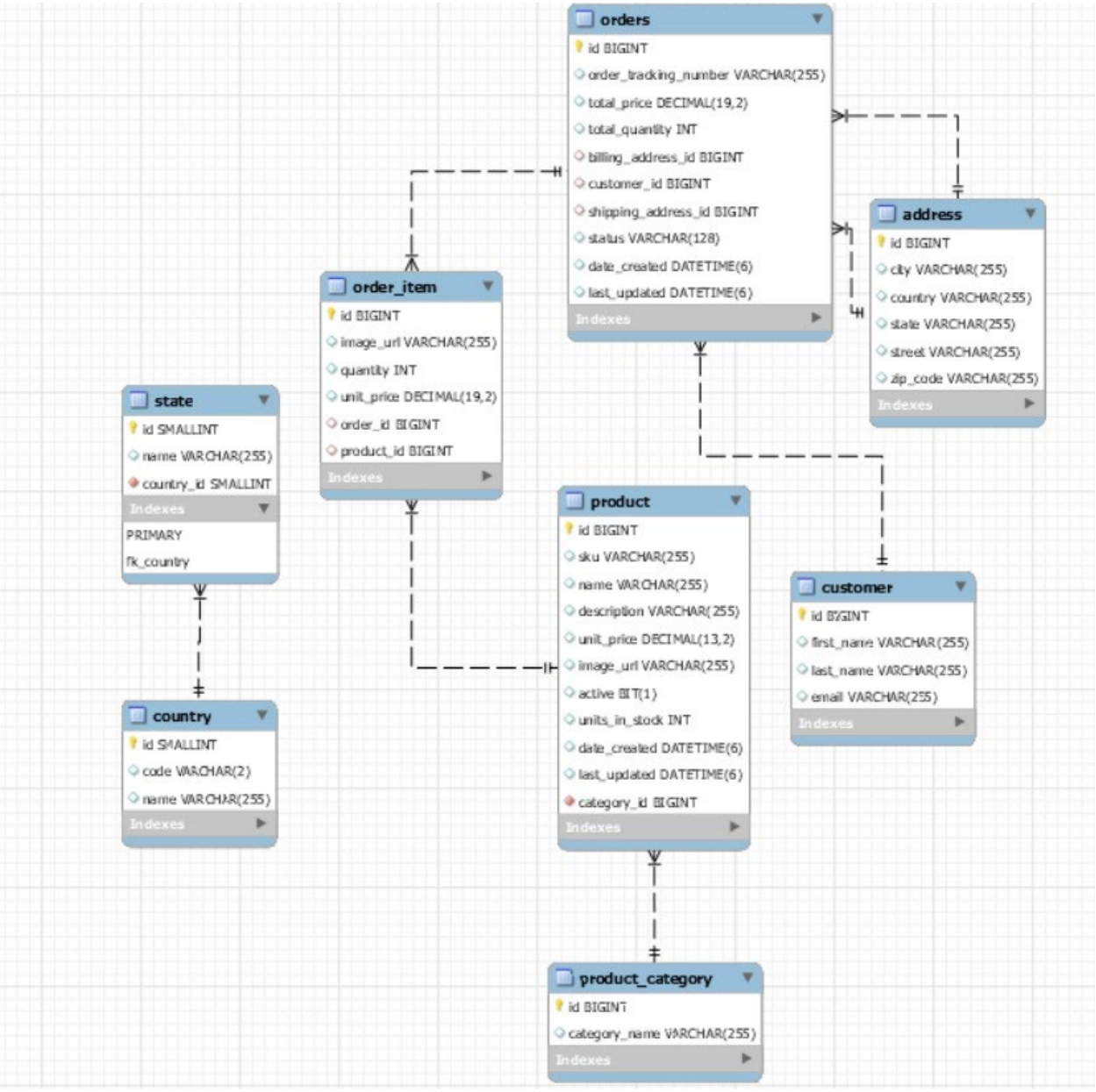
DAO's Class Diagram



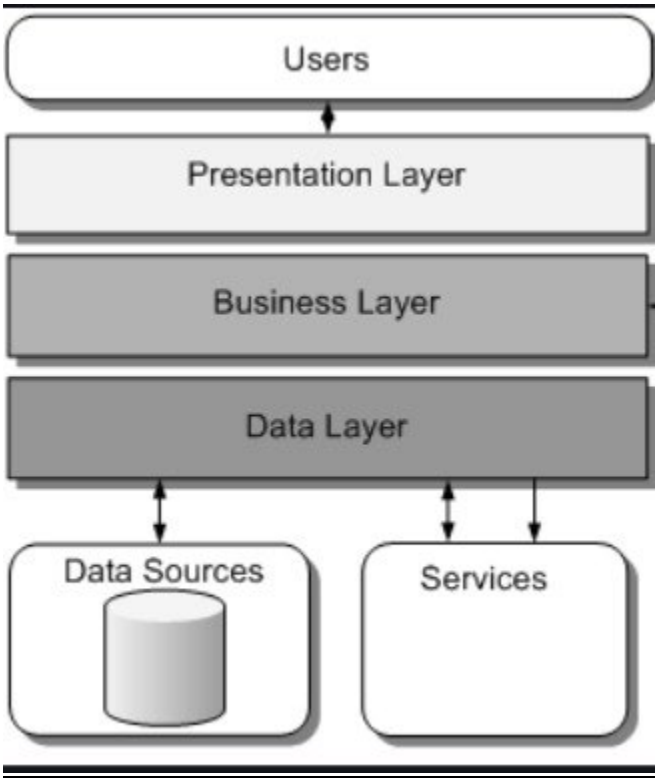
Deployment diagram


Workstation
Keyboard/monitor
Web browser
User
HTTP/HTTPS connection
Database Server
Web Server
Presentation layer
(web interface)
TCP/IP
or local socket
MySQL database
Database interface

Database



System architecture diagram



- **Code**

Backend

Java classes:

MyDataRestConfig

```
package com.se.ecommerce.config;

import com.se.ecommerce.entity.Country;
import com.se.ecommerce.entity.Product;
import com.se.ecommerce.entity.ProductCategory;
import com.se.ecommerce.entity.State;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.rest.core.config.RepositoryRestConfiguration;
```

```
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;
import org.springframework.http.HttpMethod;
import org.springframework.web.servlet.config.annotation.CorsRegistry;

import javax.persistence.EntityManager;
import javax.persistence.metamodel.EntityType;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

    private EntityManager entityManager;

    @Autowired
    public MyDataRestConfig(EntityManager entityManager){
        this.entityManager = entityManager;
    }

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config, CorsRegistry cors) {
        HttpMethod[] theUnsupportedActions = {HttpMethod.PUT, HttpMethod.POST, HttpMethod.DELETE};




        //disable HTTP methods for Product: PUT, POST and DELETE
        disableHttpMethods(Product.class, config, theUnsupportedActions);

        //disable HTTP methods for ProductCategory: PUT, POST and DELETE
        disableHttpMethods(ProductCategory.class, config, theUnsupportedActions);
```

```
        //disable HTTP methods for Country: PUT, POST and DELETE
        disableHttpMethods(Country.class, config, theUnsupportedActions);

        //disable HTTP methods for State: PUT, POST and DELETE
        disableHttpMethods(State.class, config, theUnsupportedActions);

        exposeIds(config);
    }

    private void disableHttpMethods(Class theClass, RepositoryRestConfiguration config, HttpMethod[]
theUnsupportedActions) {
        config.getExposureConfiguration()
                .forDomainType(theClass)
                .withItemExposure((metadata, httpMethods) -> httpMethods.disable(theUnsupportedActions))
                .withCollectionExposure((metada, httpMethods) -> httpMethods.disable(theUnsupportedActions));
    }

    private void exposeIds(RepositoryRestConfiguration config){
        Set<EntityType<?>> entities = entityManager.getMetamodel().getEntities();
        List<Class> entityClasses = new ArrayList <>();

        for(EntityType tempEntityType : entities){
            entityClasses.add(tempEntityType.getJavaType());
        }

        Class[] domainTypes = entityClasses.toArray(new Class[0]);
        config.exposeIdsFor(domainTypes);
    }

}
```

CheckoutController

```
package com.se.ecommerce.controller;

import com.se.ecommerce.dao.CustomerRepository;
import com.se.ecommerce.dto.Purchase;
import com.se.ecommerce.dto.PurchaseResponse;
import com.se.ecommerce.dto.SimpleCustomer;
import com.se.ecommerce.service.CheckoutService;
import org.springframework.hateoas.CollectionModel;
import org.springframework.hateoas.EntityModel;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin("http://localhost:4200")
@RestController
@RequestMapping("/api/checkout")
public class CheckoutController {

    private CheckoutService checkoutService;

    public CheckoutController(CheckoutService checkoutService) {
        this.checkoutService = checkoutService;
    }

    @PostMapping("/purchase")
    public PurchaseResponse placeOrder(@RequestBody Purchase purchase) {

        PurchaseResponse purchaseResponse = checkoutService.placeOrder(purchase);
        return purchaseResponse;
```

```
    }


}
```

CountryRespository

```
package com.se.ecommerce.dao;

import com.se.ecommerce.entity.Country;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.CrossOrigin;

@CrossOrigin("http://localhost:4200")
@RepositoryRestResource(collectionResourceRel = "countries", path="countries")
public interface CountryRepository extends JpaRepository<Country, Integer> {

}
```

CostumerRepsository

```
package com.se.ecommerce.dao;

import com.se.ecommerce.entity.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
```

```
@RepositoryRestResource(collectionResourceRel = "Customer", path = "customers")
public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

ProductCategoryRepository

```
package com.se.ecommerce.dao;

import com.se.ecommerce.entity.ProductCategory;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.CrossOrigin;

@CrossOrigin("http://localhost:4200")
@RepositoryRestResource(collectionResourceRel = "productCategory", path = "product-category")
public interface ProductCategoryRepository extends JpaRepository<ProductCategory, Long> {
}
```

ProductRepository

```
package com.se.ecommerce.dao;

import com.se.ecommerce.entity.Product;
import org.springframework.data.domain.Page;
```

```
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestParam;

@CrossOrigin("http://localhost:4200") //accept calls from web browser scripts
public interface ProductRepository extends JpaRepository<Product, Long> {

    //query method: match by category id & REST will automatically expose that endpoint
    Page<Product> findByCategoryId(@RequestParam("id") Long id, Pageable pageable);

    Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);
}
```

StateRespository

```
package com.se.ecommerce.dao;

import com.se.ecommerce.entity.State;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.CrossOrigin;

import java.util.List;

@CrossOrigin("http://localhost:4200")
@RepositoryRestResource
public interface StateRepository extends JpaRepository<State, Integer> {
```

```
    List<State> findByCountryCode(@Param("code") String code);
}
```

Purchase

```
package com.se.ecommerce.dto;

import com.se.ecommerce.entity.Address;
import com.se.ecommerce.entity.Customer;
import com.se.ecommerce.entity.Order;
import com.se.ecommerce.entity.OrderItem;
import lombok.Data;

import java.util.Set;

@Data
public class Purchase {

    private Customer customer;
    private Address shippingAddress;
    private Address billingAddress;
    private Order order;
    private Set<OrderItem> orderItems;
}
```

PurchaseResponse

```
package com.se.ecommerce.dto;

import lombok.Data;

@Data
public class PurchaseResponse {

    private final String orderTrackingNumber;

}
```

Address

```
package com.se.ecommerce.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Table(name="address")
@Getter
@Setter
public class Address {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="street")
    private String street;

    @Column(name="city")
    private String city;

    @Column(name="state")
    private String state;

    @Column(name="country")
    private String country;

    @Column(name="zip_code")
    private String zipCode;

    @OneToOne
    @PrimaryKeyJoinColumn
    private Order order;
}
```

Country

```
package com.se.ecommerce.entity;
```

```
import lombok.Getter;
import lombok.Setter;
import net.minidev.json.annotate.JsonIgnore;

import javax.persistence.*;
import java.util.List;

@Entity
@Table(name="country")
@Getter
@Setter
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="code")
    private String code;

    @Column(name="name")
    private String name;

    @OneToMany(mappedBy = "country")
    @JsonIgnore
    private List<State> states;

}
```

Costumer

```
package com.se.ecommerce.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name="customer")
@Getter
@Setter
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    @Column(name="email")
    private String email;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
```

```
    private Set<Order> orders = new HashSet<>();

    public void add(Order order) {
        if(order != null) {
            if(orders == null) {
                orders = new HashSet<>();
            }
            orders.add(order);
            order.setCustomer(this);
        }
    }
}
```

Order

```
package com.se.ecommerce.entity;

import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.*;
import java.math.BigDecimal;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
```

```
@Entity
@Table(name="orders")
@Getter
@Setter
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="order_tracking_number")
    private String orderTrackingNumber;

    @Column(name="total_quantity")
    private int totalQuantity;

    @Column(name="total_price")
    private BigDecimal totalPrice;

    @Column(name="status")
    private String status;

    @Column(name="date_created")
    @CreationTimestamp
    private Date dateCreated;

    @Column(name="last_updated")
    @UpdateTimestamp
    private Date lastUpdated;
```

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "order")
    private Set<OrderItem> orderItems = new HashSet<>();

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "shipping_address_id", referencedColumnName = "id")
    private Address shippingAddress;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "billing_address_id", referencedColumnName = "id")
    private Address billingAddress;

    public void add(OrderItem item) {
        if(item != null) {
            if(orderItems == null) {
                orderItems = new HashSet<>();
            }

            orderItems.add(item);
            item.setOrder(this);
        }
    }
}
```

OrderItem

```
package com.se.ecommerce.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.math.BigDecimal;

@Entity
@Table(name="order_item")
@Getter
@Setter
public class OrderItem {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="imageUrl")
    private String imageUrl;

    @Column(name="unit_price")
    private BigDecimal unitPrice;

    @Column(name="quantity")
    private int quantity;

    @Column(name="product_id")
    private Long productId;
```

```
    @ManyToOne
    @JoinColumn(name="order_id")
    private Order order;
}
```

Product

```
package com.se.ecommerce.entity;

import lombok.Data;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.*;
import java.math.BigDecimal;
import java.util.Date;

@Entity
@Table(name = "product")
@Data
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @ManyToOne
```

```
    @JoinColumn(name = "category_id", nullable = false)
    private ProductCategory category;

    @Column(name = "sku")
    private String sku;

    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "unit_price")
    private BigDecimal unitPrice;

    @Column(name = "image_url")
    private String imageUrl;

    @Column(name = "active")
    private boolean active;

    @Column(name = "units_in_stock")
    private int unitsInStock;

    @Column(name = "date_created")
    @CreationTimestamp
    private Date dateCreated;

    @Column(name = "last_updated")
    @UpdateTimestamp
    private Date lastUpdated;
}
```

ProductCategory

```
package com.se.ecommerce.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.util.Set;

@Entity
@Table(name = "product_category")
@Getter
@Setter
public class ProductCategory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "category_name")
    private String categoryName;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "category")
    private Set<Product> products;
}
```

State

```
package com.se.ecommerce.entity;

import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Table(name="state")
@Data
public class State {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="name")
    private String name;

    @ManyToOne
    @JoinColumn(name="country_id")
    private Country country;
}
```

CheckoutService

```
package com.se.ecommerce.service;

import com.se.ecommerce.dto.Purchase;
import com.se.ecommerce.dto.PurchaseResponse;

public interface CheckoutService {

    PurchaseResponse placeOrder(Purchase purchase);
}
```

CheckoutServiceImpl

```
package com.se.ecommerce.service;

import com.se.ecommerce.dao.CustomerRepository;
import com.se.ecommerce.dto.Purchase;
import com.se.ecommerce.dto.PurchaseResponse;
import com.se.ecommerce.entity.Customer;
import com.se.ecommerce.entity.Order;
import com.se.ecommerce.entity.OrderItem;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.Set;
import java.util.UUID;

@Service
public class CheckoutServiceImpl implements CheckoutService {

    private CustomerRepository customerRepository;

    @Autowired
    public CheckoutServiceImpl(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

    @Override
    @Transactional
    public PurchaseResponse placeOrder(Purchase purchase) {

        //retrieve the order info from dto
        Order order = purchase.getOrder();

        //generate tracking number
        String orderTrackingNumber = generateOrderTrackingNumber();
        order.setOrderTrackingNumber(orderTrackingNumber);

        //populate order with orderItems
        Set<OrderItem> orderItems = purchase.getOrderItems();
        orderItems.forEach(item -> order.add(item));

        //populate order with billingAddress and shippingAddress
```

```
        order.setBillingAddress(purchase.getBillingAddress());
        order.setShippingAddress(purchase.getShippingAddress());

        //populate costumer with order
        Customer customer = purchase.getCustomer();
        customer.add(order);

        //save to the database
        customerRepository.save(customer);

        //return a response
        return new PurchaseResponse(orderTrackingNumber);
    }

    private String generateOrderTrackingNumber() {
        //generate a UUID ( Universally Unique IDentifier)
        return UUID.randomUUID().toString();
    }
}
```

AuthentificationBean

```
package com.se.ecommerce;

public class AuthenticationBean {

    private String message;

    public AuthenticationBean(String message) {
```

```
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    @Override
    public String toString() {
        return String.format("HelloWorldBean [message=%s]", message);
    }
}
```

EcommerceApplication

```
package com.se.ecommerce;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EcommerceApplication {

            public static void main(String[] args) {
                        SpringApplication.run(EcommerceApplication.class, args);
```

```
        }

}
```

Frontend

CartDetailsComponent

```
import { Component, OnInit } from '@angular/core';
import { CartItem } from 'src/app/common/cart-item';
import { CartService } from 'src/app/services/cart.service';

@Component({
  selector: 'app-cart-details',
  templateUrl: './cart-details.component.html',
  styleUrls: ['./cart-details.component.css']
})
export class CartDetailsComponent implements OnInit {

  cartItems: CartItem[] = [];
  totalPrice: number = 0;
  totalQuantity: number = 0;
```

```
  constructor(private cartService: CartService) { }

  ngOnInit(): void {
    this.listCartDetails();
  }

  listCartDetails() {

    //get a handle to the cart items
    this.cartItems = this.cartService.cartItems;

    // subscribe to the cart totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    // subscribe to the cart totalQuantity
    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );

    //compute cart total price and quantity
    this.cartService.computeCartTotals();
  }

}
```

CartStatusComponent

```
import { Component, OnInit } from '@angular/core';
import { CartService } from 'src/app/services/cart.service';

@Component({
  selector: 'app-cart-status',
  templateUrl: './cart-status.component.html',
  styleUrls: ['./cart-status.component.css']
})
export class CartStatusComponent implements OnInit {

  public totalPrice: number = 0.00;
  public totalQuantity: number = 0.00;

  constructor(private cartService: CartService) { }

  ngOnInit(): void {
    this.updateCartStatus();
  }

  updateCartStatus() {
    //subscribe to the cart totalPrice
    this.cartService.totalPrice.subscribe(
      data => this.totalPrice = data
    );

    this.cartService.totalQuantity.subscribe(
      data => this.totalQuantity = data
    );
  }

}
```

CheckoutComponent

```
import { FactoryTarget } from '@angular/compiler';
import { error } from '@angular/compiler/src/util';
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { Router } from '@angular/router';
import { CartItem } from 'src/app/common/cart-item';
import { Country } from 'src/app/common/country';
import { Order } from 'src/app/common/order';
import { OrderItem } from 'src/app/common/order-item';
import { Product } from 'src/app/common/product';
import { Purchase } from 'src/app/common/purchase';
import { State } from 'src/app/common/state';
import { CartService } from 'src/app/services/cart.service';
import { CheckoutService } from 'src/app/services/checkout.service';
import { ShopFormService } from 'src/app/services/shop-form.service';

@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 cartService: CartService,
            private shopService: ShopFormService,
            private checkoutService: CheckoutService,
            private router: Router) { }

ngOnInit(): void {

  this.reviewCartDetails();

  this.checkoutFormGroup = this.formBuilder.group({
    costumer: this.formBuilder.group({
      firstName: [''],
      lastName: [''],
      email: ['']
    }),
    shippingAddress: this.formBuilder.group({
      street: [''],
      city: [''],
      state: [''],
      country: [''],
      zipCode: ['']
    }),
    billingAddress: this.formBuilder.group({
```

```
    street: [''],
    city: [''],
    state: [''],
    country: [''],
    zipCode: ['']
  }),
  creditCardInformation: this.formBuilder.group({
    cardType: [''],
    nameOnCard: [''],
    cardNumber: [''],
    securityCode: [''],
    expirationMonth: [''],
    expirationYear: ['']
  })
});

//populate credid card months

const startMonth: number = new Date().getMonth() + 1;
console.log("startMonth: " + startMonth);

this.shopService.getCreditCardMonths(startMonth).subscribe(
  data => {
    console.log("Retrieved credit card months: " + JSON.stringify(data));
    this.creditCardMonths = data;
  }
);

this.shopService.getCreditCardYears().subscribe(
  data => {
    console.log("Retrived credit card years: " + JSON.stringify(data));
    this.creditCardYears = data;
```

```
    }
  );

  //populate countries
  this.shopService.getCountries().subscribe(
    data => this.countries = data
  );

}

reviewCartDetails() {

  // subscribe to cartService.totalQuantity
  this.cartService.totalQuantity.subscribe(
    totalQuantity => this.totalQuantity = totalQuantity
  );

  // subscribe to cartService.totalPrice
  this.cartService.totalPrice.subscribe(
    totalPrice => this.totalPrice = totalPrice
  );

}

copyShippingAddressToBillingAddress(event: any) {
  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 = [];
  }
}

onSubmit() {
  console.log("Handling the submit button");
  console.log(this.checkoutFormGroup.get('costumer')?.value);

  // set up order
  let order = new Order();
  order.totalPrice = this.totalPrice;
  order.totalQuantity = this.totalQuantity;

  // get cart items
  const cartItems = this.cartService.cartItems;

  // - short way of doing the same thingy
  let orderItems: OrderItem[] = cartItems.map(tempCartItem => new OrderItem(tempCartItem));

  // set up purchase
  let purchase = new Purchase();

  // populate purchase - customer
  purchase.customer = this.checkoutFormGroup.controls['costumer'].value;

  // populate purchase - shipping address
  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 - billing address
    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 the CheckoutService
    this.checkoutService.placeOrder(purchase).subscribe({
      next: response => {
        alert(`Your order has been received.\nOrder tracking number: ${response.orderTrackingNumber}`);

        // reset cart
        this.resetCart();

      },
      error: err => {
        alert(`There was an error: ${err.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");
}

handleMonthsAndYears() {
  const creditCardFormGroup = this.checkoutFormGroup.get('creditCardInformation');

  const currentYear: number = new Date().getFullYear();
  const selectedYear: number = Number(creditCardFormGroup?.value.expirationYear);

  //current year equals selected year ?
  let startMonth: number;

  if(currentYear == selectedYear) {
    startMonth = new Date().getMonth() + 1;
  }
  else {
    startMonth = 1;
  }

  this.shopService.getCreditCardMonths(startMonth).subscribe(
    data => {
      this.creditCardMonths = data;
    }
  )
```

```
  }

  getStates(formGroupName: string) {

    const formGroup = this.checkoutFormGroup.get(formGroupName);
    const countryCode = formGroup?.value.country.code;
    const countryName = formGroup?.value.country.name;

    this.shopService.getStates(countryCode).subscribe(
      data => {
        if(formGroupName === 'shippingAddress') {
          this.shippingAddressStates = data;
        } else {
          this.billingAddressStates = data;
        }

        // select the first state as default
        formGroup?.get('state')?.setValue(data[0]);
      }
    )
  }
}
```

ProductCategoryComponent

```
import { Component, OnInit } from '@angular/core';
import { ProductCategory } from 'src/app/common/product-category';
import { ProductService } from 'src/app/services/product.service';
```

```
@Component({
  selector: 'app-product-category-menu',
  templateUrl: './product-category-menu.component.html',
  styleUrls: ['./product-category-menu.component.css'],
})
export class ProductCategoryMenuComponent implements OnInit {
  productCategories: ProductCategory[] = [];
  constructor(private productService: ProductService) {}

  ngOnInit(): void {
    this.listProductCategories();
  }

  listProductCategories() {
    this.productService.getProductCategories().subscribe(
      data => {
        console.log('Product Categories=' + JSON.stringify(data));
        this.productCategories = data;
      }
    )
  }
}
```

ProductDetailsComponent

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { CartItem } from 'src/app/common/cart-item';
import { Product } from 'src/app/common/product';
```

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

@Component({
  selector: 'app-product-details',
  templateUrl: './product-details.component.html',
  styleUrls: ['./product-details.component.css'],
})
export class ProductDetailsComponent implements OnInit {
  product: Product = new Product();

  constructor(
    private productService: ProductService,
    private cartService: CartService,
    private route: ActivatedRoute
  ) {}

  ngOnInit(): void {
    this.route.paramMap.subscribe(() => {
      this.handleProductDetails();
    });
  }

  handleProductDetails() {
    //get the id param string and convert it to a number
    const theProductId: number = Number(this.route.snapshot.paramMap.get('id'));
    this.productService.getProduct(theProductId).subscribe(
      data => {
        this.product = data;
      }
    )
    console.log(theProductId);
```

```
    console.log(this.product);
  }

  addToCart() {
    console.log("adding " + this.product.name)
    const theCartItem = new CartItem(this.product);
    this.cartService.addToCart(theCartItem);
  }
}
```

ProductListComponent

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { CartItem } from 'src/app/common/cart-item';
import { Product } from 'src/app/common/product';
import { CartService } from 'src/app/services/cart.service';
import { ProductService } from 'src/app/services/product.service';
//import { runInThisContext } from 'vm';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list-grid.component.html',
  styleUrls: ['./product-list.component.css'],
})
export class ProductListComponent implements OnInit {
  products: Product[];
  currentCategoryId: number;
  searchMode: boolean;
```

```
  constructor(
    private productService: ProductService,
    private route: ActivatedRoute,
    private cartService: CartService
  ) {}

  ngOnInit(): void {
    this.route.paramMap.subscribe(() => {
      this.listProducts();
    });
  }

  listProducts() {
    this.searchMode = this.route.snapshot.paramMap.has('keyword');
    if (this.searchMode) {
      this.handleSearchProducts();
    } else {
      this.handleListProducts();
    }
  }

  handleListProducts() {
    //check if "id" parameter is available
    const hasCategoryId: boolean = this.route.snapshot.paramMap.has('id');

    if (hasCategoryId) {
      // get the "id" param string and convert it to a number
      this.currentCategoryId = Number(this.route.snapshot.paramMap.get('id'));
    } else {
      //no category id available -> default to category id 1
      this.currentCategoryId = 1;
    }
```

```
    this.productService
      .getProductList(this.currentCategoryId)
      .subscribe((data) => {
        this.products = data;
      });
  }

  handleSearchProducts(){

    //TODO!
    const theKeyword  = this.route.snapshot.paramMap.get('keyword');

    this.productService.searchProducts(theKeyword).subscribe(
      data => {
        this.products = data;
      }
    )

  }

  addToCart(theProduct: Product) {
    const theCartItem = new CartItem(theProduct);
    this.cartService.addToCart(theCartItem);

  }
}
```

SearchComponent

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

@Component({
  selector: 'app-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.css']
})
export class SearchComponent implements OnInit {

  constructor(private router: Router) { }

  ngOnInit(): void {
  }

  doSearch(value: string){
    console.log(`value=${value}`);
    this.router.navigateByUrl(`/search/${value}`);
  }
}
```

CartService

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Subject } from 'rxjs';
import { CartItem } from '../common/cart-item';

@Injectable({
```

```
  providedIn: 'root'
})
export class CartService {

  cartItems: CartItem[] = [];
  totalPrice: Subject<number> = new BehaviorSubject<number>(0);
  totalQuantity: Subject<number> = new BehaviorSubject<number>(0);

  constructor() { }

  addToCart(theCartItem: CartItem) {
    //check if we already have it in the cart
    if(this.cartItems.length > 0) {
      //find the item in the cart
      for(let tempCartItem of this.cartItems) {
        if(tempCartItem.id == theCartItem.id) {
          tempCartItem.quantity++;
          this.computeCartTotals();
          return;
        }
      }

    }

    this.cartItems.push(theCartItem)
    this.computeCartTotals();
  }

  computeCartTotals() {
    let totalPriceValue: number = 0;
    let totalQuantityValue: number = 0;
```

```
    for(let currentCartItem of this.cartItems) {
      totalPriceValue += currentCartItem.quantity * currentCartItem.unitPrice;
      totalQuantityValue += currentCartItem.quantity;
    }

    //publish the new values for total price and quantity
    this.totalPrice.next(totalPriceValue);
    this.totalQuantity.next(totalQuantityValue);

    // log cart data
    this.logCartData(totalPriceValue, totalQuantityValue);
  }

  logCartData(totalPriceValue: number, totalQuantityValue: number) {
    console.log('Contents of the cart');
    for(let tempCartItem of this.cartItems) {
      const subtotalPrice = tempCartItem.quantity * tempCartItem.unitPrice;
      console.log('name: ' + tempCartItem.name + ', quantity: ' + tempCartItem.quantity + ', unitPrice= ' +
tempCartItem.unitPrice);
    }
    console.log('total price ' +  totalPriceValue.toFixed(2) + " quantity " + totalQuantityValue);
  }
}
```

CheckoutService

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
```

```
import { Observable } from 'rxjs';
import { Purchase } from '../common/purchase';

@Injectable({
  providedIn: 'root'
})
export class CheckoutService {

  private purchaseUrl = "http://localhost:8080/api/checkout/purchase";

  constructor(private httpClient: HttpClient) { }

  placeOrder(purchase: Purchase): Observable<any> {
    console.log("Purchase: " + purchase.shippingAddress.country + " " + purchase.shippingAddress.state);
    return this.httpClient.post<Purchase>(this.purchaseUrl, purchase);
  }
}
```

ProductService

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Product } from '../common/product';
import { map } from 'rxjs/operators';
import { ProductCategory } from '../common/product-category';

@Injectable({
```

```
  providedIn: 'root',
})
export class ProductService {
  private baseUrl = 'http://localhost:8080/api/products';
  private categoryUrl = 'http://localhost:8080/api/product-category';

  constructor(private httpClient: HttpClient) {}

  getProductList(theCategoryId: number): Observable<Product[]> {
    const searchUrl = `${this.baseUrl}/search/findByCategoryId?id=${theCategoryId}`;
    return this.getProducts(searchUrl);
  }

  getProductCategories(): Observable<ProductCategory[]> {
    return this.httpClient
      .get<GetResponseProductCategory>(this.categoryUrl)
      .pipe(map((response) => response._embedded.productCategory));
  }

  searchProducts(theKeyword: string | null): Observable<Product[]> {
    const searchUrl = `${this.baseUrl}/search/findByNameContaining?name=${theKeyword}`;
    return this.getProducts(searchUrl);
  }

  private getProducts(searchUrl: string): Observable<Product[]> {
    return this.httpClient
      .get<GetResponseProducts>(searchUrl)
      .pipe(map((response) => response._embedded.products));
  }

  getProduct(theProductId: number): Observable<Product> {
    const productUrl = this.baseUrl + "/" + theProductId;
```

```
    return this.httpClient.get<Product>(productUrl);
  }
}

interface GetResponseProducts {
  _embedded: {
    products: Product[];
  };
}

interface GetResponseProductCategory {
  _embedded: {
    productCategory: ProductCategory[];
  };
}
```

ShopFormService

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';
import { Country } from '../common/country';
import { State } from '../common/state';
```

```
@Injectable({
  providedIn: 'root'
})
export class ShopFormService {

  private countriesUrl = 'http://localhost:8080/api/countries';
  private statesUrl = 'http://localhost:8080/api/states';

  constructor(private httpClient: HttpClient) { }

  getCountries(): Observable<Country[]> {
    return this.httpClient.get<GetResponseCountries>(this.countriesUrl)
    .pipe(map((response) => response._embedded.countries));
  }

  getStates(theCountryCode: string): Observable<State[]> {
    //search url
    // const searchStatesUrl = '${this.statesUrl}/search/findByCountryCode?code=${theCountryCode}';
    const searchStatesUrl = this.statesUrl + '/search/findByCountryCode?code=' + theCountryCode;
    return this.httpClient.get<GetResponseStates>(searchStatesUrl).pipe(
      map(response=> response._embedded.states)
    );
  }

  getCreditCardMonths(startMonth: number): Observable<number[]> {
    let data: number[] = [];

    //build array for month dropdown list
    // start from current month

    for(let theMonth = startMonth; theMonth <= 12; theMonth++) {
      data.push(theMonth);
```

```
    }

    return of(data); // wrap an onject as an obesevable
  }


  getCreditCardYears() : Observable<number[]> {
    let data: number[] = [];

    //build array for year dropdown list

    const startYear: number = new Date().getFullYear();
    const endYear: number = startYear + 10;

    for(let theYear = startYear; theYear <= endYear; theYear++) {
      data.push(theYear);
    }

    return of(data);
  }

}

interface GetResponseCountries {
  _embedded: {
    countries: Country[];
  }
}

  interface GetResponseStates {
  _embedded: {
    states: State[];
```

```
  }
}
```

AppModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { ProductListComponent } from './components/product-list/product-list.component';
import { HttpClientModule } from '@angular/common/http'
import { ProductService } from './services/product.service';
import { Routes, RouterModule } from '@angular/router';
import { ProductCategoryMenuComponent } from './components/product-category-menu/product-category-menu.component';
import { SearchComponent } from './components/search/search.component';
import { ProductDetailsComponent } from './components/product-details/product-details.component';
import { CartStatusComponent } from './components/cart-status/cart-status.component';
import { CartDetailsComponent } from './components/cart-details/cart-details.component';
import { CheckoutComponent } from './components/checkout/checkout.component';
import { ReactiveFormsModule } from '@angular/forms';

const routes: Routes = [
  {path: 'checkout', component: CheckoutComponent},
  {path: 'cart-details', component: CartDetailsComponent},
  {path: 'products/:id', component: ProductDetailsComponent},
  {path: 'search/:keyword', component: ProductListComponent},
  {path: 'category/:id', component: ProductListComponent},
```

```
  {path: 'category', component: ProductListComponent},
  {path: 'products', component: ProductListComponent},
  {path: '', redirectTo: '/products', pathMatch: 'full'},
  {path: '**', redirectTo: '/products', pathMatch: 'full'}
];

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductCategoryMenuComponent,
    SearchComponent,
    ProductDetailsComponent,
    CartStatusComponent,
    CartDetailsComponent,
    CheckoutComponent,
  ],
  imports: [
    RouterModule.forRoot(routes),
    BrowserModule,
    HttpClientModule,
    ReactiveFormsModule
  ],
  providers: [ProductService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Address

```
export class Address {
    street: string;
    city: string;
    state: string;
    country: string;
    zipCode: string;
}
```

CartItem

```
import { Product } from "./product";

export class CartItem {
    id: number;
    name: string;
    imageUrl: string;
    unitPrice: number;
    quantity: number;

    constructor(product: Product) {
        this.id = product.id;
        this.name = product.name;
        this.imageUrl = product.imageUrl;
        this.unitPrice = product.unitPrice;
```

```
        this.quantity = 1;
    }
}
```

Country

```
export class Country {
    id: number;
    name: string;
    code: string;
}
```

Customer

```
export class Customer {
    firstName: string;
    lastName: string;
    email: string;
}
```

OrderItem

```
import { CartItem } from "./cart-item";

export class OrderItem {
    imageUrl: string;
    unitPrice: number;
    qunatity: number;
    productId: number;

    constructor(cartItem: CartItem) {
        this.imageUrl = cartItem.imageUrl;
        this.unitPrice = cartItem.unitPrice;
        this.qunatity = cartItem.quantity;
        this.productId = cartItem.id;
    }
}
```

Order

```
export class Order {
    totalQuantity: number;
    totalPrice: number;
}
```

ProductCateogory

```
export class ProductCategory {
   id!: number;
   categoryName!: string;
}
```

Product

```
export class Product {
  id: number;
  sku: string;
  name: string;
  description: string;
  unitPrice: number;
  imageUrl: string;
  active: boolean;
  unitsInStock: number;
  dateCreated: Date;
  lastUpdate: Date;
  quantity: number;

}
```

Purchase

```
import { Address } from "./address";
import { Customer } from "./customer";
import { Order } from "./order";
import { OrderItem } from "./order-item";

export class Purchase {
    order: Order;
    customer: Customer;
    shippingAddress: Address;
    billingAddress: Address;
    orderItems: OrderItem[];
}
```

State

```
export class State {
    id: number;
    name: string;
}
```