HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INTERNATIONAL EDUCATION

COURSE NAME: Web Programming

FINAL PROJECT REPORT

Project name:

COMPUTER HARDWARE RETAIL WEB APPLICATION

Instructor: Ms. Mai Anh Thơ

Course ID: WEPR330479E_23_1_02FIE

Group: 5

Period: 1st Sem/2023-2024

Ho Chi Minh city, Dec __, 2023

(This page was intentionally left blank)

LIST OF STUDENTS – GROUP 5

Project: Computer hardware retail store (PC Parts Shop)

ID	Full name	Role
21110066	Phạm Vũ Bảo Nhân	- Team leader - Report/Presentation - Concept art - Data creation - Front-end design & developer - Back-end design & developer - Tester/Debugging

Instructor's comments

Ho Chi Minh city, Dec __, 2023

Grading

Table of Contents

CHAPTER I: INTRODUCTION 5
1. Specifications 5
2. Technical specifications 5
3. Concept designs 6
4. Finished product 7
CHAPTER II: BUSINESS LAYER 11
1. Overview 11
2. Class diagram of business objects 11
3. The Customer class 12
CHAPTER III: CONTROLLER LAYER 13
1. Model-View-Controller (MVC) diagram 13
2. The CatalogController class 13
CHAPTER IV: PROJECT STRUCTURE 16
1. Directory structure 16
2. Description 16
CHAPTER V: DATABASE 17
1. Conceptual level database design 17
2. Logical level database design 17
CHAPTER VI: DATA LAYER 18
1. Class diagram of data access classes 18
2. The ProductDB class 18
CHAPTER VII: SCRIPTS 22
1. The use of JavaScript 22
2. Swiper JS 22
3. Cleave.js 22
4. Other scripts 22

CHAPTER I: INTRODUCTION

1. Specifications

1.1. Problem statement/Use case

An online store will perform the following functions:

- Customer, Account creation and manipulation
- Sign up/Login/Sign out functions for Accounts
- Display information about Products
- Search/Filter by Name/Type for Products and display them in a Catalog
- Cart to save Products for easy access
- Checkout functionality (For displaying Invoices)
- Payment system (Not implemented in this application)

1.2. Overview

PC Parts Shop website is a mock-up website for a fictional computer hardware retailing company. This web application will allow users to navigate between several webpages using Controllers (Or Servlets).

Users (Or Customers) can look up information about any desired Product, proceed to add them to a Cart associated with an Account that they have created.

Once the Customers are satisfied with the Items they bought, they can go to checkout and pay for the Products.

2. Technical specifications

2.1. Technologies used

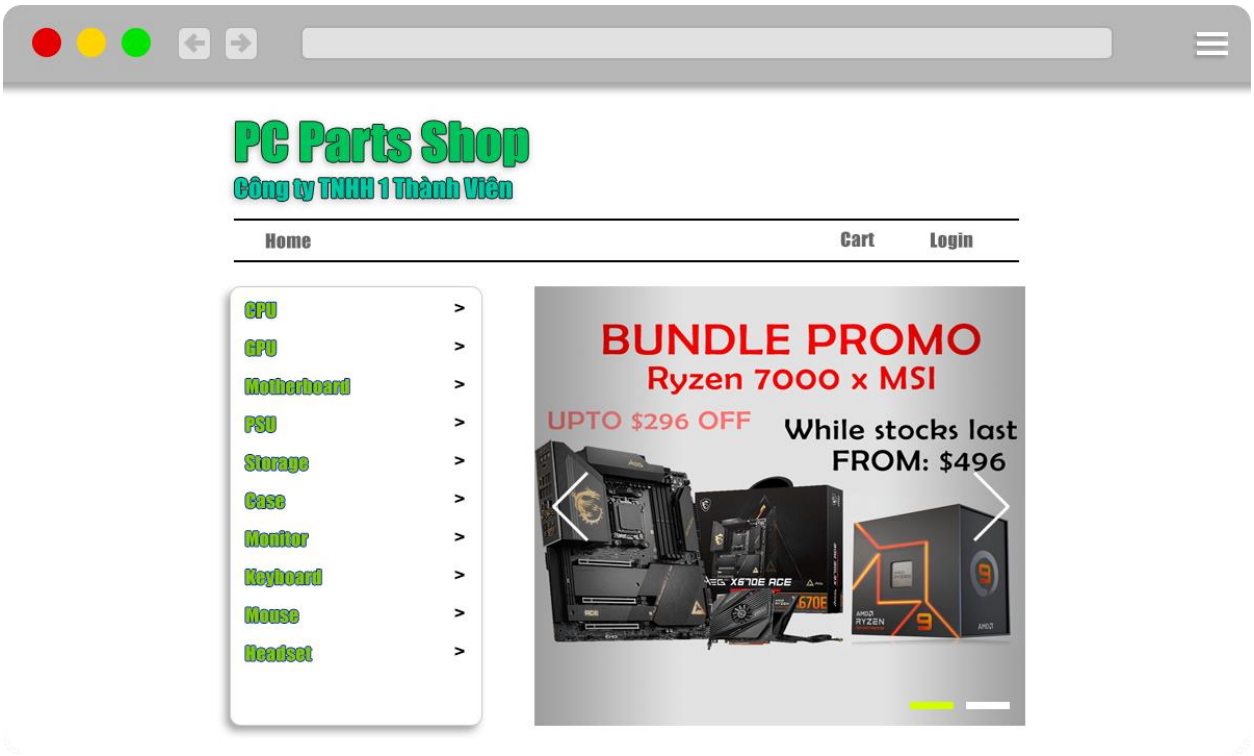
- HTML/CSS
- Java, JavaScript/TypeScript
- Apache Maven 3.9
- Apache Tomcat 8.5
- MySQL Database

2.2. External libraries

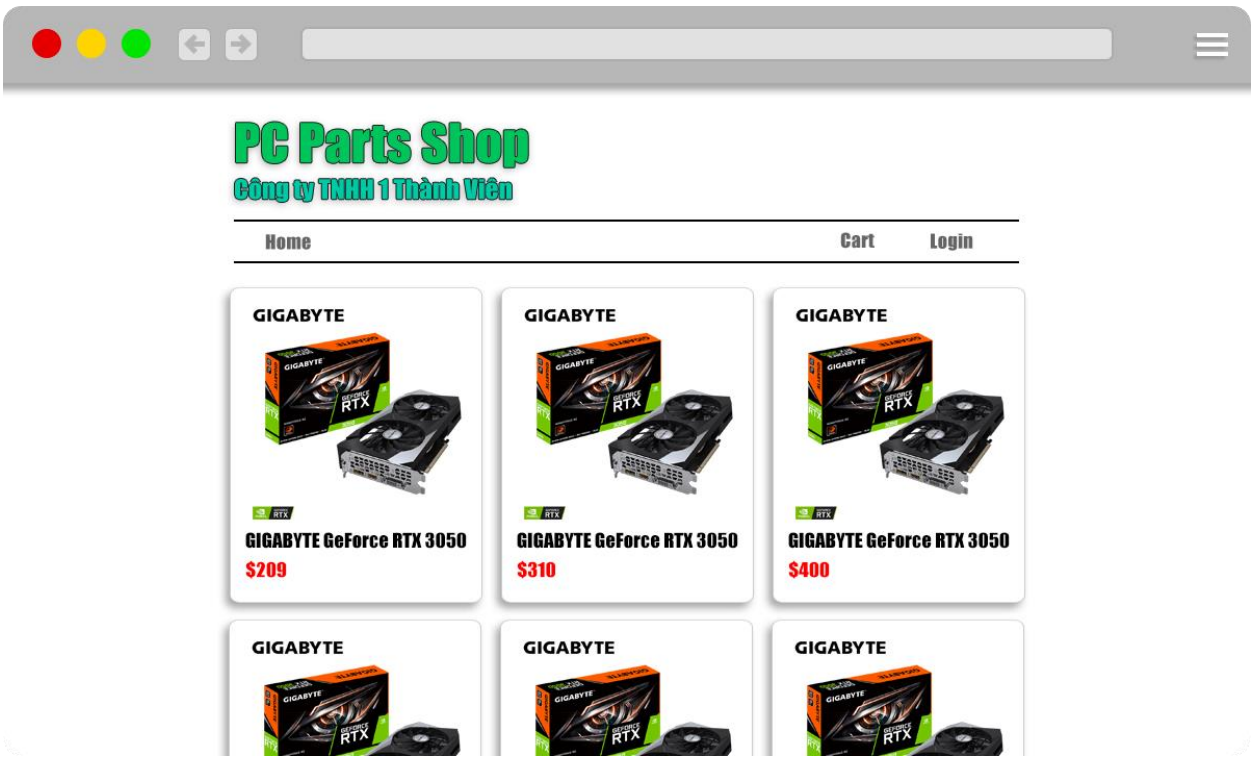
- Java Standard Tag Library (JSTL)
- Java Server Pages API (JSP)
- Connector/J (Java Database Connectivity, for MySQL)
- EclipseLink (Java Persistence API, for object-relational database mapping)
- Swiper JS (Animations)
- Cleave.js (Form input formatting)

3. Concept designs

Concept design of Home.jsp



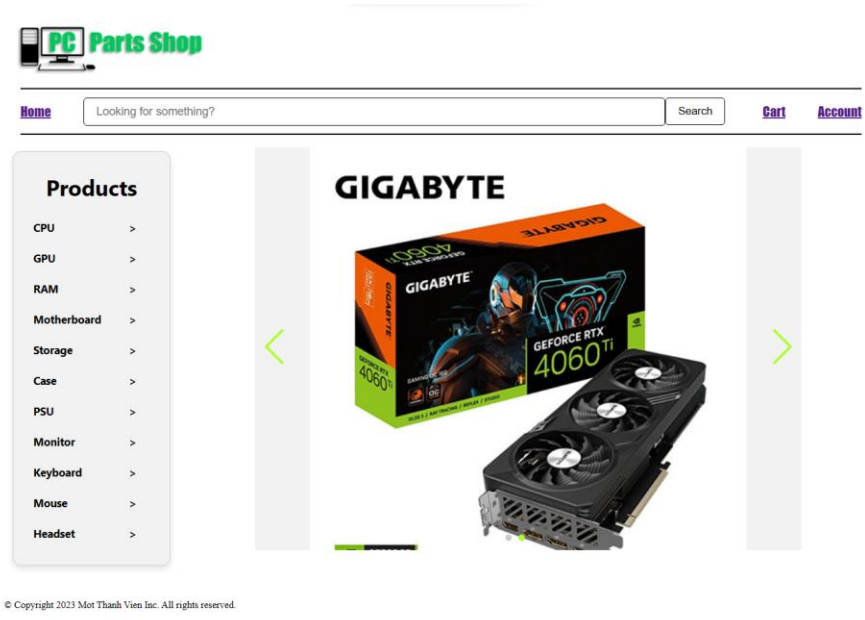
Concept design of Catalog.jsp



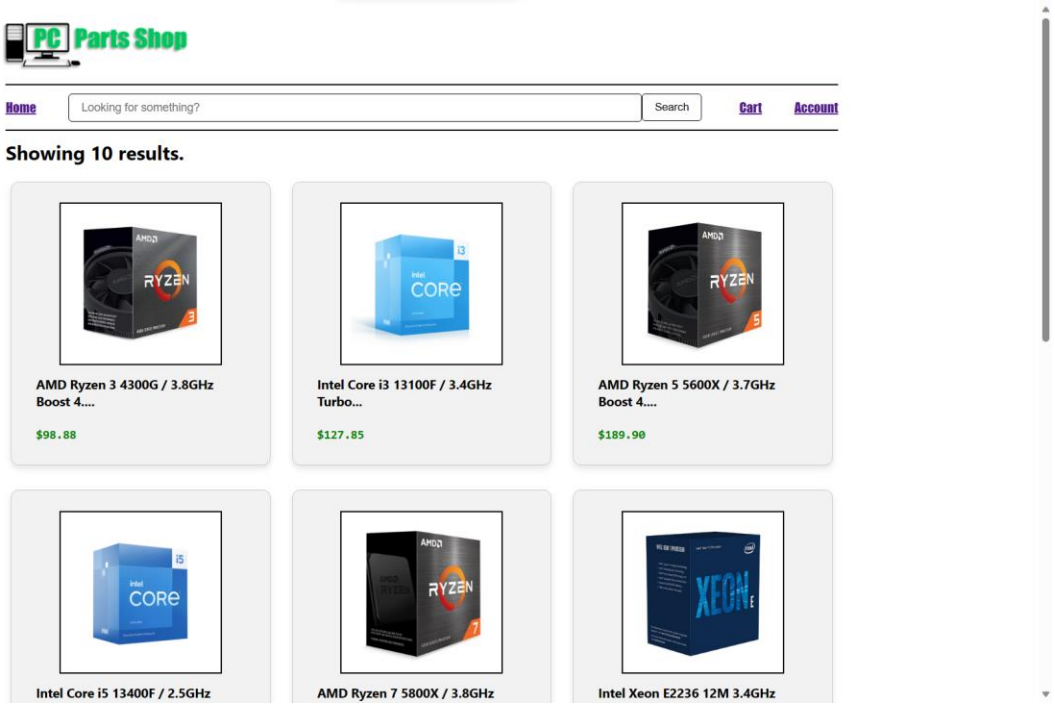
Concept design of Product.jsp

4. Finished product

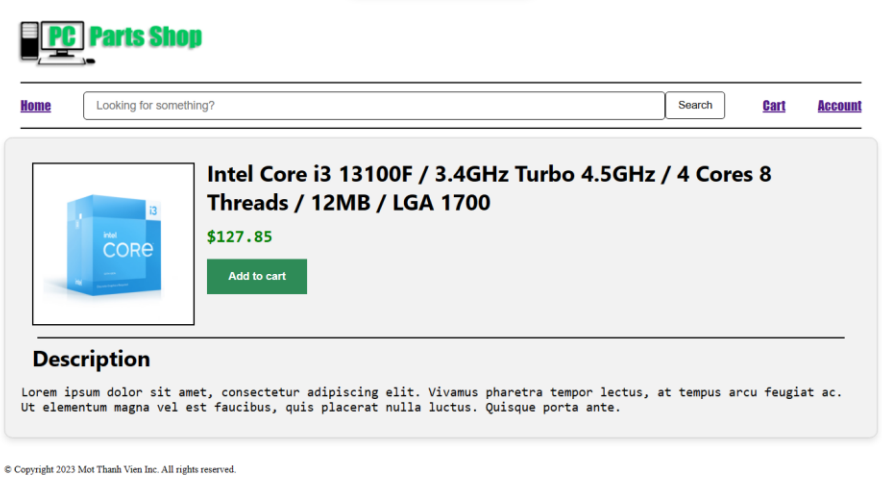
Home.jsp



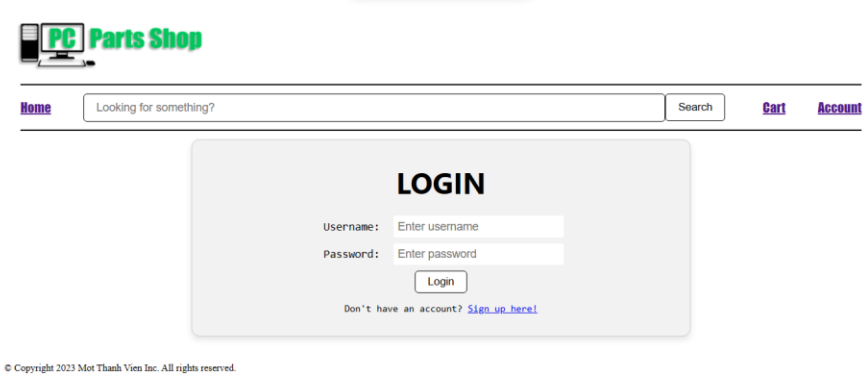
Catalog.jsp



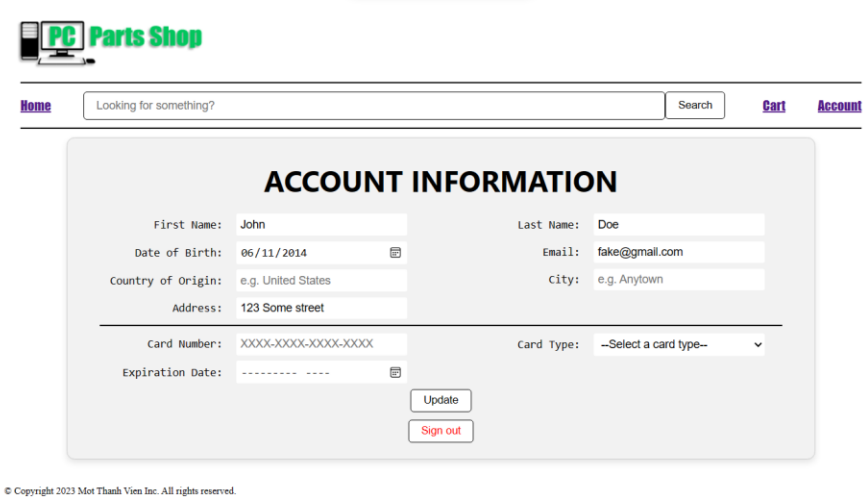
Product.jsp



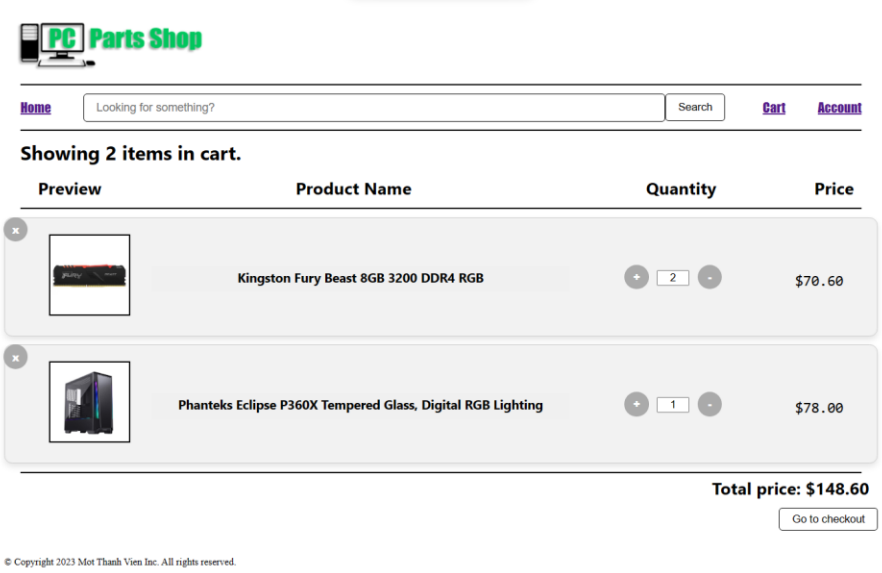
Login.jsp



Account.jsp



Cart.jsp



Invoice.jsp

CHAPTER II: BUSINESS LAYER

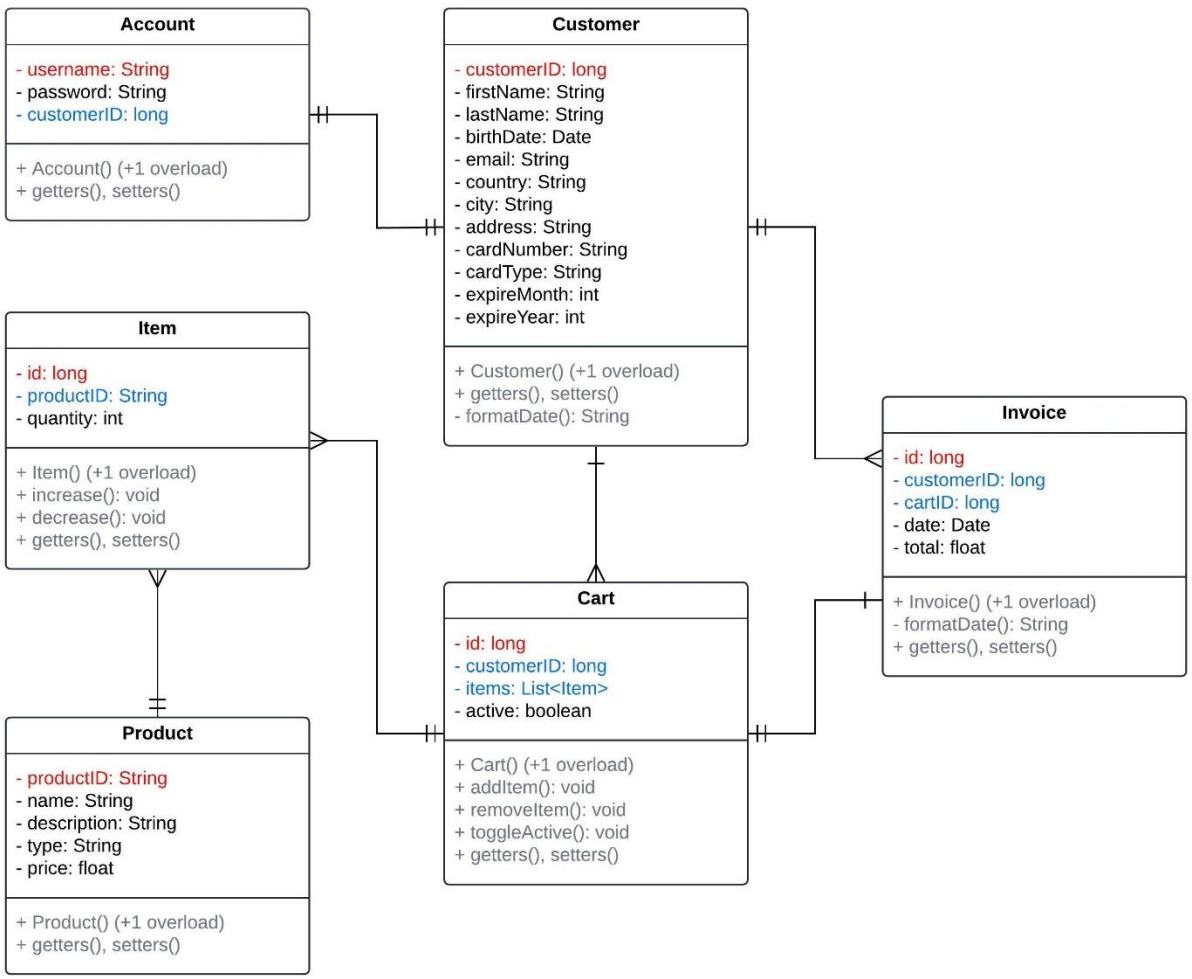
1. Overview

After a thorough analysis of the use case for this web application, it was decided that there would be at least six business objects:

- Customer (Stores customer information)
- Account (Provides access for different customers)
- Product (Stores information about products)
- Item (Stores a singular product and its quantity within a cart)
- Cart (Contains a list of items awaiting purchase, belongs to one customer)
- Invoice (Contains customer information and cart information)

2. Class diagram of business objects

The class diagram below will provide a brief description of what attributes and methods each Class will contain.



3. The Customer class

3.1. JPA integration

Java Persistence API uses "annotations" to declare special Entity classes to map to its corresponding database entity in MySQL. The implementation of JPA within a web application such as this will make a developer's job easier as they do not have to worry about designing a database and instead, focus on creating the application.

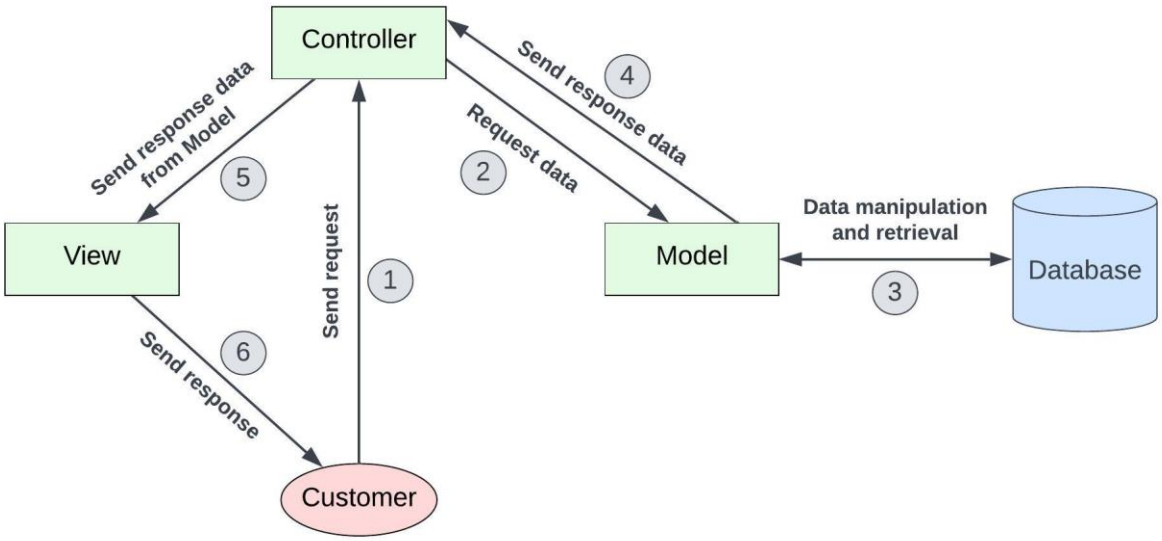
3.2. Code snippet

```
@Entity
public class Customer {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long customerID;

	private String firstName;
	private String lastName;
	private Date birthDate;
	private String email;
	private String country;
	private String city;
	private String address;

	private String cardNumber;
	private String cardType;

	@Transient
	private String cardExpire;

	private int expireMonth;
	private int expireYear;

	@Transient
	private String formattedBirthDate;

	@OneToOne(mappedBy = "customer")
	private Account account;

	// Constructor
	public Customer() {…}

	public Customer(String firstName, String lastName, Date birthDate,
String email, String country, String city, String address, String cardNumber,
String cardType, String cardExpire) {…}

	// Method
	private String formatDate(Date date) {
		SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
		return format.format(date);
	}

	// Getters and Setters
```

CHAPTER III: CONTROLLER LAYER

1. Model-View-Controller (MVC) diagram

The MVC diagram below depicts how all the Controller classes function within the web application.



2. The CatalogController class

2.1. The use of Java Servlets

Servlets are used in web development to handle requests and generate dynamic web content. They are Java classes that follow the request-response programming model and are commonly used to extend web server applications.

Some of their functions include:

- Handling HTTP requests (Process GET, POST requests, get parameters,…)
- Dynamic content generation (Generate HTML, XML,…)
- Session management (Tracking and maintenance of user sessions)
- Form processing (Extract form data, validate input,…)

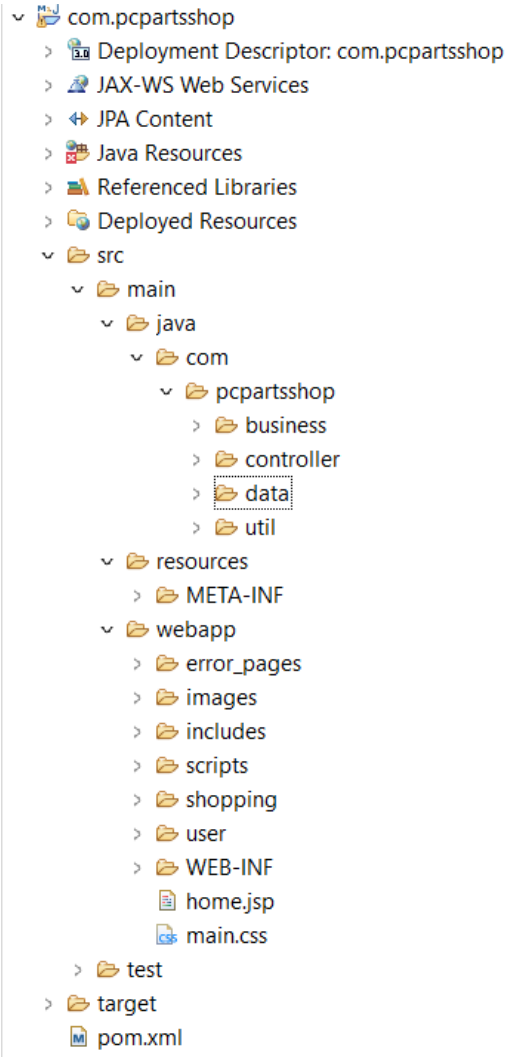
2.2. Code snippet

```
package com.pcpartsshop.controller;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

import com.pcpartsshop.data.ProductDB;
import com.pcpartsshop.util.SQLUtil;

@WebServlet("/catalogHandler")
public class CatalogController extends HttpServlet {
	private static final long serialVersionUID = 1L;

	@Override
	public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		String requestURI = request.getRequestURI();
		String action = null;
		String url = null;

		if (requestURI.endsWith("/catalogHandler")) {
			action = request.getParameter("action");
			if (action == null || action.equals("")) {
				url = displayAll(request, response);
			}
			if (action.equals("search")) {
				url = searchCatalog(request, response);
			} else if (action.equals("filter")) {
				url = filterCatalog(request, response);
			} else {
				url = "/error_pages/error_404.jsp";
			}
		}
		getServletContext().getRequestDispatcher(url).forward(request,
response);
	}

	@Override
	public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		doGet(request, response);
	}
```

```
	private String displayAll(HttpServletRequest request,
HttpServletResponse response) {
		String url = null;

		request.setAttribute("title", "All products");
		int productCount = ProductDB.getTotalCount();
		request.setAttribute("productCount", productCount);

		String catalogResult =
SQLUtil.getProductCatalog(ProductDB.selectAllProducts());
		request.setAttribute("catalogResult", catalogResult);
		url = "/catalog";

		return url;
	}
	private String filterCatalog(HttpServletRequest request,
HttpServletResponse response) {
		String type = request.getParameter("type");
		String url = null;

		if (type == null || type.equals("")) {
			return displayAll(request, response);
		}
		request.setAttribute("title", "Category: " + type);
		int productCount = ProductDB.getCountOfType(type);
		request.setAttribute("productCount", productCount);

		String catalogResult =
SQLUtil.getProductCatalog(ProductDB.filterByType(type));
		request.setAttribute("catalogResult", catalogResult);
		url = "/catalog";

		return url;
	}
	private String searchCatalog(HttpServletRequest request,
HttpServletResponse response) {
		String q = request.getParameter("q").trim();
		String url = null;

		if (q == null || q.equals("")) {
			return displayAll(request, response);
		}
		request.setAttribute("title", "You searched for: " + q);
		int productCount = ProductDB.getCountOfName(q);
		request.setAttribute("productCount", productCount);

		String catalogResult =
SQLUtil.getProductCatalog(ProductDB.filterByName(q));
		request.setAttribute("catalogResult", catalogResult);
		url = "/catalog";

		return url;
	}
}
```

CHAPTER IV: PROJECT STRUCTURE

1. Directory structure



2. Description

- All JSP files are stored in the webapp directory, in each of their sub-directories
- The web.xml file is stored in the webapp/WEB-INF directory
- The persistence.xml file is stored in the resources/META-INF directory
- The Java classes, including Servlets, are stored in their respective sub-directories of the Java package that corresponds to its name
- The external libraries required by the application are stored in the WEB-INF/libs sub-directory
- All images used by the application are stored in the images sub-directory of webapp

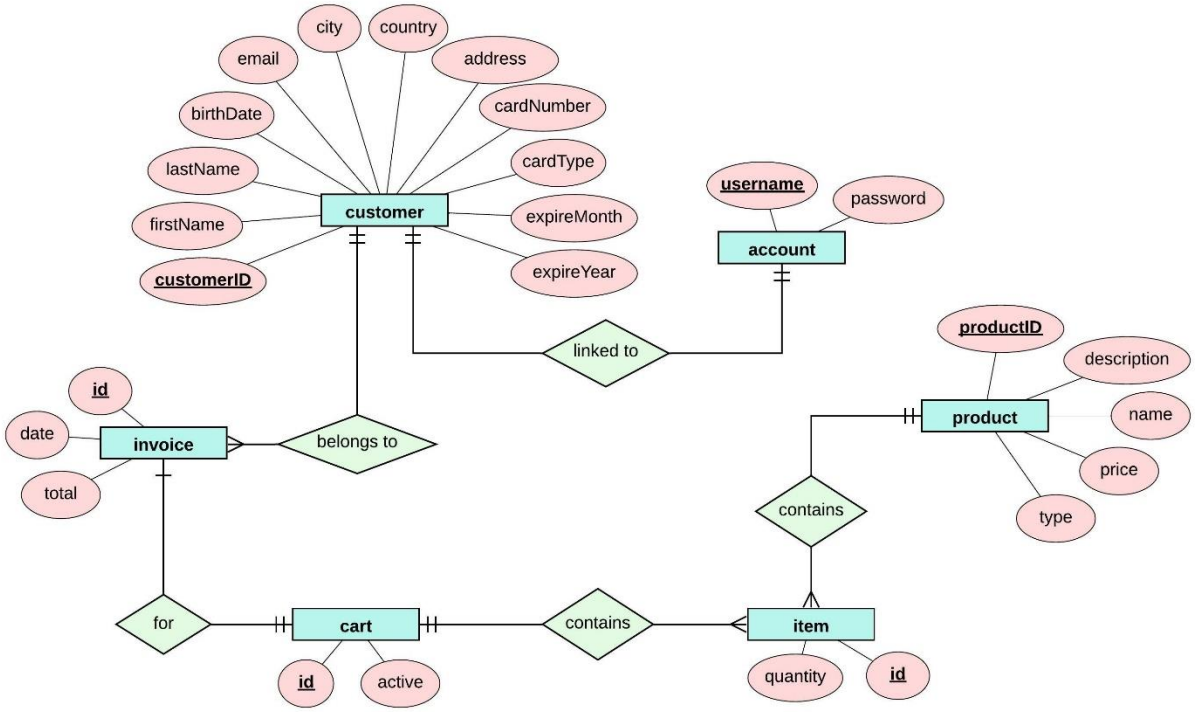
CHAPTER V: DATABASE

1. Conceptual level database design

1.1. Overview

The following Entity Relationship Diagram shows the diagram for the PC Parts Shop database. This diagram shows that this database stores most of its data in six tables that correspond to six of the business objects.

1.2. Entity Relationship Diagram (ERD)



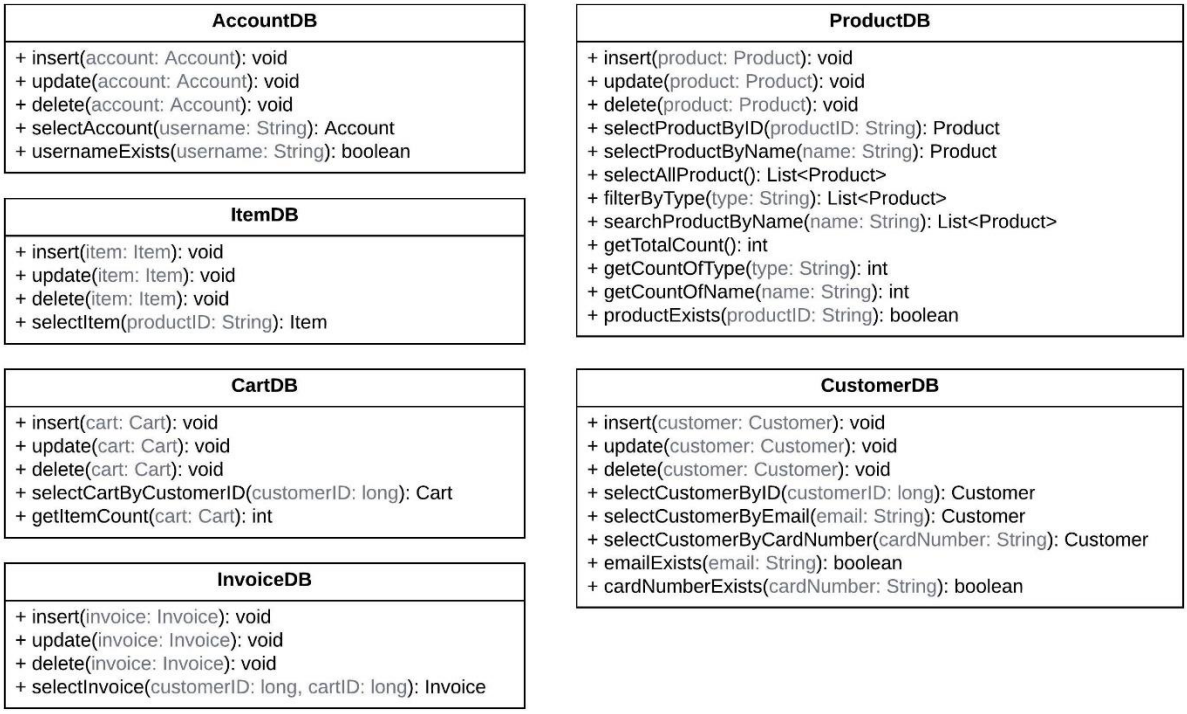
2. Logical level database design

From the ERD, we have:

1. Customer(customerID, firstName, lastName, birthDate, email, city, country, address, cardNumber, cardType, expireMonth, expireYear)
2. Account(username, password, customerID)
3. Cart(id, customerID, active)
4. Product(productID, name, description, type, price)
5. Item(id, quantity, productID)
6. Cart_Item(cart_ID, item_ID)
7. Invoice(id, date, total, customerID, cartID)

CHAPTER VI: DATA LAYER

1. Class diagram of data access classes



2. The ProductDB class

2.1. DBUtil class

Another Java class that plays an important role in retrieving data from the database is the DBUtil class.

```
public class DBUtil {
	private static final EntityManagerFactory emf =
Persistence.createEntityManagerFactory("pc_parts_shopPU");

	public static EntityManagerFactory getEmFactory() {
		return emf;
	}
}
```

The reason why this class plays a significant role in this web application is due to the fact that it creates an EntityManager object initialized by the persistence.xml file to form a connection and create a means to extract and insert data to the database.

2.2. Code snippet

```
public class ProductDB {
	public static void insert(Product product) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		EntityTransaction transaction = em.getTransaction();

		transaction.begin();
		try {
			em.persist(product);
			transaction.commit();
		} catch (Exception ex) {
			System.out.println(ex);
			transaction.rollback();
		} finally {
			em.close();
		}
	}

	public static void update(Product product) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		EntityTransaction transaction = em.getTransaction();

		transaction.begin();
		try {
			em.merge(product);
			transaction.commit();
		} catch (Exception ex) {
			System.out.println(ex);
			transaction.rollback();
		} finally {
			em.close();
		}
	}

	public static void delete(Product product) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		EntityTransaction transaction = em.getTransaction();

		transaction.begin();
		try {
			em.remove(em.merge(product));
			transaction.commit();
		} catch (Exception ex) {
			System.out.println(ex);
			transaction.rollback();
		} finally {
			em.close();
		}
	}
```

```
	public static Product selectProductByID(String productID) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		String queryString = "SELECT p FROM Product p WHERE p.productID
= :id";
		TypedQuery<Product> query = em.createQuery(queryString,
Product.class);
		query.setParameter("id", productID);

		Product product = null;
		try {
			product = query.getSingleResult();
		} catch (NoResultException ex) {
			System.out.println(ex);
		} finally {
			em.close();
		}
		return product;
	}

	public static Product selectProductByName(String name) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		String queryString = "SELECT p FROM Product p WHERE p.name
= :name";
		TypedQuery<Product> query = em.createQuery(queryString,
Product.class);
		query.setParameter("name", name);

		Product product = null;
		try {
			product = query.getSingleResult();
		} catch (NoResultException ex) {
			System.out.println(ex);
		} finally {
			em.close();
		}
		return product;
	}

	public static List<Product> selectAllProducts() {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		String queryString = "SELECT p FROM Product p";
		TypedQuery<Product> query = em.createQuery(queryString,
Product.class);

		List<Product> productList = new ArrayList<>();
		try {
			productList = query.getResultList();
		} catch (NoResultException ex) {
			System.out.println(ex);
		} finally {
			em.close();
		}
		return productList;
	}
```

```
	public static List<Product> filterByType(String type) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		String queryString = "SELECT p FROM Product p WHERE p.type
= :type";
		TypedQuery<Product> query = em.createQuery(queryString,
Product.class);
		query.setParameter("type", type);

		List<Product> productList = new ArrayList<>();
		try {
			productList = query.getResultList();
		} catch (NoResultException ex) {
			System.out.println(ex);
		} finally {
			em.close();
		}
		return productList;
	}

	public static List<Product> searchProductByName(String name) {
		EntityManager em = DBUtil.getEmFactory().createEntityManager();
		String queryString = "SELECT p FROM Product p WHERE p.name LIKE
CONCAT('%', :name, '%')";
		TypedQuery<Product> query = em.createQuery(queryString,
Product.class);
		query.setParameter("name", name.toLowerCase());

		List<Product> productList = new ArrayList<>();
		try {
			productList = query.getResultList();
		} catch (NoResultException ex) {
			System.out.println(ex);
		} finally {
			em.close();
		}
		return productList;
	}

	public static int getTotalCount() {
		List<Product> productList = selectAllProducts();
		return productList.size();
	}
	public static int getCountOfType(String type) {
		List<Product> productList = filterByType(type);
		return productList.size();
	}
	public static int getCountOfName(String name) {
		List<Product> productList = filterByName(name);
		return productList.size();
	}
	public static boolean productExists(String productID) {
		Product product = selectProductByID(productID);
		return product != null;
	}
```

CHAPTER VII: SCRIPTS

1. The use of JavaScript

If you have noticed, this web application also makes use of some JavaScript libraries such as Swiper JS and Cleave.js to perform some complex tasks on the webpages.

To simplify the functions of what these libraries do:

- Swiper JS handles the animations of the automatically scrolling element on the home.jsp file
- Cleave.js handles the formatting of <input> fields inside HTML <form> tags

You can use either Node.js TypeScript to program these scripts or look up their documentation and manually program them.

2. Swiper JS

```
const swiper = new Swiper('.swiper', {
	autoplay: {
		delay: 3000,
		disableOnInteraction: false,
	},
	loop: true,

	pagination: {
		el: '.swiper-pagination',
		clickable: true,
	},

	navigation: {
		nextEl: '.swiper-button-next',
		prevEl: '.swiper-button-prev',
	},
});
```

3. Cleave.js

```
document.addEventListener('DOMContentLoaded', function() {
	var cardNumber = document.getElementById('creditCardInput');
	var cleave = new Cleave(cardNumber, {
		creditCard: true,
		delimiter: '-',
	});
});
```

4. Other scripts

4.1. Script for going back a page

This script is used in the error pages (Error 404, 400, Java error). When clicked, rather than having to go back to Home page, the users will navigate back to the page right before the error occurred.

```
function goBack() {
	window.history.back();
}
```

4.2. Script for product filtering

This script is used in home.jsp page in the Category tab on the left. When clicked, the script will:

- Take the value of the clicked button
- Attach to the “type” <input> field
- Submit the parent form that will call the CatalogController class

```
function submitForm(value) {
	document.getElementById('type').value = value;
	document.getElementById('filter').submit();
}
```

4.3. Miscellaneous

You will notice that there are 2 files present in the scripts folder:

- product-insert.txt (Contains MySQL queries to insert data)
- run-commands.bat (Batch file to run all queries in product-insert.txt)

These 2 files are mainly used to simplify the process of inputting data for Product table inside “pc_parts_shop” table, in case any data was lost.

Because there aren’t programs to handle the input of the data above, this project utilizes a more tedious approach to inserting data (Manually typing out queries, run a file that will run all queries).

(This page was intentionally left blank)