# EECS 4413, Fall 2023: team Project Specifications -- V1

## Teams

This is a teamwork project. The size of the team is required to be minimum 2 and maximum 4 students per team. Ideally a team should be comprised of 3 students. You should have formed your team and registered your team information on eClass by Oct 16.

## Basic functionalities

In this project you are going to create an online store for a retail company. Your job involves designing, implementing, testing and deploying a web application that supports Customers and Administrators to interact with the e-store.

Customers can execute several use cases, *including*

- Register (sign up), Sign in, sign out

- List catalogue items
- Sort items by prices, names
- Filter catalog items by categories, brand, model
- View details (description, brand, price etc) of a product

- Add items to shopping cart
- Edit or remove items from the shopping cart
- "Check out" by providing credit card information and shipping information to purchase the items in the shopping cart

- Check purchase history
- etc

Administrators: are the owners of the store; they have access to the following use case

- Check sales history.
- Check inventory
- Maintain user account
- etc

You are welcome to add more features, some examples are given in the Executive Summary part later in this document.

## Development and Runtime Frameworks

You will develop a multi-tier web application, with clear separation between front and back-end.

You are allowed to use **any** technology presented in class, and are also encouraged to go beyond the APIs covered in lectures and Labs. For example, you can use **Java Spring or Sprint Boots** for back-end content, use **React** or **JS/Angular** for front end, use **Node.js** frameworks for both front and back ends. (Some tutorials on the topics will be provided).

You can also use databases other than MySQL and SQLite, e,g., PostgreSQL and even Mongodb.

It is expected that each member of the team can contribute to and understand each part of the project and therefore agrees on the technologies used in the project.

## Design and implementation

It is important that the architecture of your store exhibits:

- Clear separation between front and back-end
- MVC design with a clear separation of business logic and presentation. Web API that mediates between front- and back-end,
- DAO design pattern.
- Other architecture and Design patterns when appropriate (Factory, Singleton, Observer etc.)
- Testcases (based on curl) for the back-end
- Good coding style (modularity, comments, readability, etc.)
- Robustness to user inputs

Below are the *SUGGESTED* main components/services of your e-store. It is acceptable to deviate from this as long as you justify it in the design document. So, your architecture may comprise more components/services than the ones reported below.

## Back-end and Model services

**A. Data Access**
This component mediates between your application's business logic and the data base(s). It should be scalable and configurable.

**B. Catalog Component/Service**
The Catalog provides the APIs and the implementation to support use cases related to the products sold by the estore

**C. Ordering Component/Service**
The Ordering provides the APIs and the implementation of use cases than manage user orders

**D. Identity management component/service.** Provides the interfaces and the implementation for user registration, log in and log out.

**E. Shopping cart component/service.** Provides services for managing the shopping basket, including the management of state

## Web APIs (this is the set of API exposed by the back-end)

**F. Controller/API Gateway**
The component/service mediates between the "model" and the "views".

## Front-end components

The front end can be built as a collection of dynamic views. Below are the main views, use them as examples:

**G. Catalog View**
Displays the contents of the store organized by category and by product. The visitor must be able to

1. browse the catalog and see All, By Brand, By Categories products
2. select an item and see the detailed information for that item (description, brand, price etc.).
3. add an individual item to shopping cart.

**H. Shopping Cart View**

The Shopping Cart Page allows a visitor to review the order

1. view all items in the shopping cart and their information (price, etc.).
2. remove individual items from the shopping cart or increase/decrease the quantity. While doing so, the total bill is updated.
3. Button or link that allow user to go back to continue shopping.
4. "Checkout" submit button indicating they wish to purchase the items in the shopping cart.

**I. Checkout View**

1. either log into their account with a password, or create a new account to check out.
2. for a new account they enter their account name, password, and default billing and shipping information. The new account is submitted to the Order Processing service.
3. to submit their order, they verify their billing and shipping information, and enter in their credit card number.
4. Confirm order" button

*Note*: You do not need to use a 3rd party payment service for this project, create a simple payment web service that mimics a real payment. Design a simple algorithm to accept or deny a payment request. E.g., deny every 3rd request of payment, or other dummy algorithms . If the order is approved, you should display "Order Successfully Completed." If it is denied, you should display "Credit Card Authorization Failed.", and allow user the enter again or quit.

**J. Registration View.** Provides the user interface for registering/maintaining an account.

1. log in
2. log out
3. Register

**K. Administrator/Analytics Page View**

The Administrator should be able to

1. view the items sold

## Executive summary

You can use any tools you choose. You are free to design the interface and business logic.
Listed below are required features (in red), and examples of extra features (in blue)

**Here are required functionalities:**

- Design and implementation:
  - Use MVC design pattern with a clear separation of business logic and presentation. Web API that mediates between front- and back-end,
  - Use DAO design pattern

- Customers can execute several use cases, *including*
  - Register (sign up), entering customer info (name, address, etc)
  - Sign in
  - See account page with customer info, purchase history etc.
  - Sign out

- List catalogue items
    - Items should have images
- Sort items by prices (sort items based on ascending prices or based on the descending prices)
- Sort items by names (i.e., in the alphabetical order)
- Filter catalog items by categories
- Filter catalog items by brand
- View details of a product
    - Should have inventory information (quantity remaining)

- Add items to shopping cart
- Edit or remove items from the shopping cart
- Go back to continue shopping.
- "Check out" by providing credit card information and shipping information to purchase the items in the shopping cart

- Administrators: are the owners of the store; they have access to the following use case
    - Check sales history.
    - Maintain customer account pages with customer info, purchase history etc
    - Maintain inventory, to change quantity of product (add inventory)

**List above are required features.**
You are welcome to add more features. Extra features could get bonus marks. Listed below are some suggested example features (you are not limited to these features. You are encouraged to be creative in designing your project):

- **Design and implementation**
    - Use design patterns (in addition to MVC and DAO which are required), e.g,. Singleton, Factory, Observer, Façade etc
    - Has one or more **Web services** that follow REST API (e.g., payment as a separate Web Service).

**Customer**:
- Search product by type, keyword etc..
- Read and write reviews on items and rate items using five stars
- Ask questions to a live chatbot able to answer basic questions
- send confirmation email to customer after checkout.
- Show 'featured/hot/recommended' products, based on sales
- For videos, with trailer.
- For glasses/clothes/wig, with virtual "try on" on customer uploaded pictures
- For books, with preview
- Encypted login information
- …. Other creative features… TBA

**Administrator**:
- Can run a Reports on sales with option to save as disk file (e.g. as PDF files).
- Authorize/approve rejected payments
- …. Other creative features… TBA

## Team development and deployment

By the end of the term (**Around Nov 6, TBD**)

- Your e-commerce application is expected to be deployed in cloud, including the database, at a public address (e.g., AWS, Google cloud or the web host). That is, the application is runnable by visiting a public address
- Or, on Docker so it can be run by the instructor and TA on other machines.

## Main deliverables

- By the end of the term (**Around Aug 8, TBD**), you will present your application.
- By the end of the term, you will submit
  - A project design report (template will be provided soon)
    - should be 10~ 15 pages more details to be added
    - rationale of design choices, design details, class diagrams, database design, contribution of member, challenges, creative features etc
    - more details to be provided.
  - Docker file or link to the application if deployed in the cloud.
    - **One team member should focus more on deployment. Explore use of Docker, or ways to upload to cloud host**. (Other members focus on front end and back end each)
  - Source code of program.
    - If you use cloud GitHub, provide the link to it and give your TA access to it. Otherwise, provide your source files as jar/war file. The TA should see your code and scripts.

**Appendix:**

**Sample SQL schemas for e-store database**

Some sample data schemas are attached at the end of this document. You are welcome to use any free database from Cloud (SQL or noSQL). *You need to extend the schema and add more content to the tables.*

-use it as a starting example

- you will need to adapt them for other DBMS
- you will need to extend them as you see fit. For instance, each item may have more attributes, for example, for glasses, may have clour, a shape (e.g., oval, round, square), a size, and a weight (in grams).
- you will need more tables. For example, you may need table for **review, billing_address** etc..

```
/* create Item/product table*/
CREATE TABLE Item(
 itemID  VARCHAR(20)  NOT  NULL  PRIMARY  KEY,
 name VARCHAR(60) NOT NULL,
 description VARCHAR(60) NOT   NULL,
 category VARCHAR(60) NOT NULL,
 brand VARCHAR(60) NOT NULL,
 quantity INT NOT  NULL,
 price INT NOT NULL
 …..
);

/*example insert data into item table*/
INSERT INTO ITEM (ItemID, name, description, category, brand, price, quantity) VALUES ('b001',
'Little Prince', 'a book for all ages', 'book', 'penguin', 20, 100);
INSERT INTO ITEM (ItemID, name, description, category, brand, price, quantity) VALUES ('c001',
'iPad', 'a portable device for personal use', 'computer', 'Apple', 500, 100);
INSERT INTO ITEM (ItemID, name, description, category, brand, price, quantity) VALUES
('d001', 'laptop', 'a laptop for personal use', 'Dell',  1500, 100);

/*create an address table*/
CREATE TABLE Customer (
 id   INT NOT NULL,
 firstName VARCHAR(100) NOT NULL,
 lastName VARCHAR(20) NOT NULL,
 addressID INT  NOT NULL,
 …..

 PRIMARY KEY(id)
 FOREIGN KEY (addressID) REFERENCES Address (id)
);

/*create an address table*/
CREATE TABLE Address ( id     INT NOT NULL,
 street VARCHAR(100) NOT NULL,
 province VARCHAR(20) NOT NULL,
 country VARCHAR(20) NOT NULL,
```

```
 zip     VARCHAR(20) NOT NULL,
 phone   VARCHAR(20),
 …..
 PRIMARY KEY(id),
);

/*populate the address table*/

INSERT INTO Address (id, street, province, country, zip, phone) VALUES (1, '567 Yonge St',
'ON', 'Canada', 'K1E 6T5' ,'647-123-4567');
INSERT INTO Address (id, street, province, country, zip, phone) VALUES (2, '945 Avenue
rd', 'ON', 'Canada', 'M1C 6K5' ,'416-123-8569');
INSERT INTO Address (id, street, province, country, zip, phone) VALUES (3, '189 Keele
St.', 'ON', 'Canada', 'K3C 9T5' ,'416-123-9568');


/* create Purchase Order (PO) table */  used for viewing sales history

/* Purchase Order
CREATE TABLE PO
 ( id INT NOT NULL,
 customerID VARCHAR(20) NOT NULL,
 itemID  VARCHAR(20) NOT NULL,
 dateOfPurchase  VARCHAR(20) NOT NULL,
 ….

 PRIMARY KEY(id),
 FOREIGN KEY (customerID) REFERENCES Customer (id)
 FOREIGN KEY (itemID) REFERENCES Item (id)

);
```