Team Project Specifications : EECS 4413

Executive summary

A convenience company wants to create an online store. You must design, implement, test and deploy a web application that supports Customers and Administrators to interact with the e-store.

Registered Customers can execute several use cases, *including*

- List catalogue items
- Filter catalog items by type
- Filter catalog items by brand
- View details 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
- Write reviews on items
- Register
- Sign in
- Sign out

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

- Run Reports on sells
- Run reports on application usage

You will develop a multi-tier web application, with clear separation between front and back-end. *It is expected you add an original use case to the above list.*

Development and Runtime Frameworks

You are allowed to use any technology presented in class and encouraged to go beyond the APIs covered in Labs and lectures. For example, you can use Java Spring for back-end content and JS/Angular frameworks for front end. 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.

Team development and deployment

An ideal application will be deployed like this:

- Your e-commerce application should be deployed in cloud, at a public address.
- The database runs in cloud
- On eClass you provide the design document, testcases, a link to the application.
- 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.

A sample data schema is 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.*

Design and implementation

It is important that the architecture of your store exhibits:

- Clear separation between front and backed
- A Web API that mediates between front- and back-end
- Testcases (based on curl) for the backend
- Good coding style (modularity, comments, readability, etc.)
- Architecture and Design patterns (MVC, Observer, DAO, etc.)
- Robustness to user inputs
- Quality attributes (performance, security)

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.

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 out.

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

F. Analytics component/service. Provides statistics about the application and its usage

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

E. 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 or as SPA (Single Page Application). Below are the three 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

- UC M1: browse the catalog and see All, By Brand, By Type products
- UC M2: select an item and see the information for that item (price, ratings etc.).
- UC M3: add a review for an item
- UC M5: add an individual item to shopping cart.
- UC M6: Shopping cart submit button

H. Shopping Cart View
The Shopping Cart Page allows a visitor to review the order

- UC C1: view all items in the shopping cart and their information (price, etc.).
- UC C2: remove individual items from the shopping cart or increase/decrease the quantity. While doing so, the total bill is updated.
- UC C3: "Checkout" submit button indicating they wish to purchase the items in the shopping cart.

I. Checkout View

- UC P1: either log into their account with a password, or create a new account.
- UC P2: 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.
- UC P3: to submit their order, they verify their billing and shipping information, and enter in their credit card number.
- UC P4: "Confirm order" button

Note: You do not need to use a 3rd party payment service for this project, create a simple payment service that mimics a real payment in this way: You honour two consecutive requests, but you deny every 3rd request of payment. If the order is approved, you should display "Order Successfully Completed." If it is denied, you should display "Credit Card Authorization Failed."

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

- UC-R1 log in
- UC-R2 log out
- UC R: Register

K. Administrator/Analytics Page View
The Administrator should be able to

- UC A1: generate a report with the items sold each month
- UC A2: provide reports on the website usage (see VisitEvent table)

Non-functional requirements

M. Performance and Scalability. Conduct a performance test of your application. Choose one of your backend API. Test your application with 1, 2…N clients. Draw the throughput and the response time curves. N should be chosen such that utilization of the server is less than 60%. Assume 3 seconds "think time" between user requests. You can use a load testing program such as JMeter. Record the time the thread sends the request, record the time the thread gets back the response; the difference is the response time. Repeat that for 1, 2…N threads. Check the slides for the general shape of the response time…By response time we mean the average across all clients(threads)

N. Security
The store website should run under https, SSL. *SSL is activated by setting up the application server, you do not have to program anything special in your application.*

- "Checkout" page and "Confirm Order" action must be secured so that a login is required and the password is not passed in plain text.
- The visitor MUST type in their credit card each time. It should not be stored.
- The application should be tested for two common vulnerabilities: SQL injection and cross-site scripting.
    - Document one testcase for each threat (see the design document).

Deliverables

A single zip file containing the following (in a reasonable folder hierarchy) main components

a) A Design document
b) A link to your deployed application in the cloud.
c) Your source war file or a link to the git repository (private)

The **design document** should be less than 10 pages, excluding the front page and table of content, and contain

- a front page with the title, team members, team member contributions: Detail the individual contributions in one paragraph/member; also explain how each team member learned about elements of the projects done by other members. Each member of the team signs the document to attest that team member contributions reflect the reality.
- a table of content
- An Architecture section, in which the overall architecture, system components and 2 sequence diagrams (for 2 use cases) are presented. One use case should describe your original use case
- A Design section where you describe 3 patterns you used in the project. Also, you explain the main design decisions, trade-offs. Provide the class diagram for one component.
- Implementation section. Here describe the implementation decisions, the trade-offs. Also, discuss the limitations, especially with regard to testing.
- Security testing report. Explain how you tested for security vulnerability and what is not tested.
- Performance testing report.
- team member contributions: Detail the individual contributions in one paragraph/member; also explain how each team member learned about elements of the projects done by other members.

Your project (github or war) should contain

- documented and well organized source code
- SQL file used to create and populate the database tables (need to extend the schema provided below)
- A script with testcases (curl commands) for the backend APIs
- readme file explaining how to run it; also how the TA should ran the test cases.

SQL schema for e-store database

-This was tested on Derby 10
https://db.apache.org/derby/docs/10.5/ref/index.html

-use it as a starting example
-you might need to adapt it for other DMS
-you can extend it as you see fit (note it does not have user registration)

/*uncomment DROP TABLE if you want to remove the tables and reinitialize*/

```
/*
DROP TABLE VisitEvent;
DROP TABLE POItem;
DROP TABLE PO;
DROP TABLE Address;
DROP TABLE Item;
*/

/* create Item table*/

CREATE TABLE Item(
 bid     VARCHAR(20) NOT NULL PRIMARY KEY,
 name    VARCHAR(60) NOT NULL,
 description    VARCHAR(60) NOT NULL,
 type     VARCHAR(60) NOT NULL,
 brand     VARCHAR(60) NOT NULL,
 quantity    INT NOT NULL,
 price     INT NOT NULL
);

/*insert data into item table*/

INSERT INTO ITEM (bid, name, description, type, brand, price, quantity) VALUES ('b001',
'Little Prince', 'a book for all ages', 'book', 'Penguin', 20, 100);
INSERT INTO ITEM (bid, name, description, type, brand, price, quantity) VALUES ('c001', 'iPad', 'a
device for personal use', 'computer', 'Apple', 500, 100);
INSERT INTO ITEM (bid, name, description, type, brand, price, quantity) VALUES ('d001', 'laptopm',
'a device for personal use', 'computer', 'Apple', 1500, 100);


/*create an addtess 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, '123 Yonge St', 'ON',
'Canada', 'K1E 6T5' ,'647-123-4567');
INSERT INTO Address (id, street, province, country, zip, phone) VALUES (2, '445 Avenue rd', 'ON',
'Canada', 'M1C 6K5' ,'416-123-8569');
INSERT INTO Address (id, street, province, country, zip, phone) VALUES (3, '789 Keele St.', 'ON',
'Canada', 'K3C 9T5' ,'416-123-9568');


/* create Purchase Order(PO) table */

/* Purchase Order
 * lname:           last name
```

```
 * fname:             first name
 * id:                purchase order id
 * status: status of purchase: Processed, Denied, Ordered
 */

CREATE TABLE PO (
 id      INT NOT NULL,
 lname    VARCHAR(20) NOT NULL,
 fname    VARCHAR(20) NOT NULL,
 status   VARCHAR(20)NOT NULL,
 address  INT NOT NULL,
 PRIMARY KEY(id),
 FOREIGN KEY (address) REFERENCES Address (id)
);

/*
* Inserting data for table
'PO' */

INSERT INTO PO (id, lname, fname, status, address) VALUES (1, 'John', 'White', 'PROCESSED', 1);
INSERT INTO PO (id, lname, fname, status, address) VALUES (2, 'Peter', 'Black', 'DENIED',    2);
INSERT INTO PO (id, lname, fname, status, address) VALUES (3, 'Andy', 'Green', 'ORDERED',    3);


/*create table Purchase Ordert Item, contains items on each order*/



CREATE TABLE POItem (
 id      INT NOT NULL,
 bid     VARCHAR(20) NOT NULL,
 price   INT NOT NULL,
 PRIMARY KEY(id,bid),
 FOREIGN KEY(id) REFERENCES PO(id),
 FOREIGN KEY(bid) REFERENCES Item(bid)
);

/*
*Inserting data for table 'POitem'
*/

INSERT INTO POItem (id, bid, price) VALUES (1, 'b001', 20);
INSERT INTO POItem (id, bid, price) VALUES (2, 'c001', 500);


/* visit to website
 * ipaddress: varchar
 * day:               date
 * bid:   unique identifier of item
 * eventtype:       status of purchase
 */


CREATE TABLE VisitEvent (
 ipaddress varchar (20) NOT NULL,
 day       varchar(8) NOT NULL,
```

```
 bid        varchar(20) not null,
 eventtype            varchar(20) NOT NULL,
 FOREIGN KEY(bid) REFERENCES Item(bid)
);

/*
*  data for table 'VisitEvent'
*/
INSERT INTO VisitEvent (ipaddress, day, bid, eventtype) VALUES ('1.23.4.5','12202022', 'b001',
'VIEW');
INSERT INTO VisitEvent (ipaddress, day, bid, eventtype) VALUES ('1.23.4.5', '12242022',
'b001', 'CART');
INSERT INTO VisitEvent (ipaddress,day, bid, eventtype) VALUES ('1.23.4.5', '12252022', 'b001',
'PURCHASE');

/* check the content of the tables*/


SELECT * FROM VisitEvent;
SELECT * FROM POItem;
SELECT * FROM PO;
SELECT * FROM Address;
SELECT * FROM Item;
```