Project topic, LE/EECS 4413 Z

Executive summary

The brick-and-mortar company Mom&Pop wants to create an online book store. They want these types of Users: Visitors, Customers, Administrators, and Partners, to interact with the store. *You can replace book store with any type of store as long as it supports the use cases below.*

All Users can

- Log in
- Log out
- Register

Visitors and Customers can

- Browse a product catalogue of books
- Select, add, remove books to/from a shopping cart
- "Check out" by providing credit card information and shipping information to purchase the items in the shopping cart
- Search the store

Administrators: are the owners of the store

- Run Reports

Partners: are business partners, they can programmatically (through REST APIs)

- Search the Catalog and Order
- ~~Search the Catalog (text search-Google search)~~[1]

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 JavaServer Faces or Java Spring for dynamically generating content, or JS frameworks for browser programming. You will use Java and JDBC in the middle tier. It is expected that each member of the team can contribute to and understand each part of the project. Also, you are encouraged to use Maven as a build tool.

Team development and deployment

A. 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 Moodle you provide the design document, ~~the war file~~ and a link to the application.
- If you use cloud github, provide the link to it and give your TA access to it.

A sample data schema is attached at the end of this document. It is in MySQL but 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 book store exhibits:

- Good coding style (modularity, comments, readability, etc.)
- Architecture and Design patterns (MVC, Observer, DAO, etc.)
- Testing (Robustness to user inputs, Performance and Security)

Below are the SUGGESTED main components of your bookstore. It is acceptable to deviate from this as long as you justify it in the design document.

A. Data Access Component
This component mediates between your application's business logic and the data base. It should be scalable, use connectionPool and be configurable.

B. Product Catalog Component/Service
The Product Catalog is offered both as browser component and as a Web Service (REST). When offered as a service, it is just for external Partners. It should support the following functionality, where productId is the bid (ISBN) from the datatabase:

- *getProductInfo(productId) / /gets the detailed product information for a product as a JSON file. This is a REST operation.*

C. Order Process Component/Service
The Order Process is offered both as a browser action and as a Service (REST). It is offered as a service for external Partners. It should support the following:

- *getOrdersByPartNumber(productId)// return a list of all orders This is a REST service and returns JSON*

D. Session Controller
The session controller mediates between the "model" (B and C above) and the "views" (E, F, and G below). The controller manages session information related to the shopping cart (items selected, address, etc.).

E. Book Store Main Page
Displays the contents of the store organized by category and by product.
The visitor must be able to

- UC M1: browse Book Categories (Science, Engineering, Fiction) and see the books available.
- UC M2: select a book and see the information for that title (price, ratings etc.).
- UC M3: add a review for book
- UC M4: search the store
- UC M5: add an individual book to shopping cart.
- UC M6: Shopping cart button

F. Shopping Cart Page
The Shopping Cart Page allows a visitor to

- UC C1: view all the 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: "Payment" submit button indicating they wish to purchase the items in the shopping cart.

G. Payment Page
The visitor can

- 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."

H. Analytics Page

The Administrator should be able to

- UC A1: generate a report with the books sold each month
- UC A2: provide real time analytics (Listeners) with most popular products (life time), like "top 10 sold books")
- UC A3: provide annomized reports with user buying statistics. Note: by statistics, one can understand the amount each user has spent, the zip code of the buyers. By "anonymized" one can understand that the user names are replaced on the fly (in the filter) with ****, so the admin cannot identify the buyers (due to privacy concerns). You can use Filters for this.

I. Web services. Your application has two web services, as specified above, each one with one method (the italic method). The web services respond to REST messages. The REST message should reply with an XML message or JSON message using the XML schema po.xml, available here: http://www.w3.org/TR/xmlschema-0/.

J. Performance and Scalability. Conduct a performance test of your application. For this project focus just on one Service (either B or C above). 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 computer is less than 60% Assume 3 seconds "think time" between user requests. An implementation suggestion: you can emulate each client with a thread…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)

K. Security

In addition, 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.*
"Payment" 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.

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 IBM cloud.
c) Your source war file or a link to the git repository (private)
d) A war file with test code such rest client tests classes and scalability test code. Name it Team_Name_Test.war.

The **design document** should be less than 10 pages, (excluding the sample outputs and code listings, also excluding the front page and table of content) and contain

- a front page with the title, team members, team members contribution (see below)
- a table of content

- the architecture, including UML use cases, class diagrams and 2 sequence diagrams (for 2 use cases). Describe the patterns you used, the main design decisions, trade-offs
- implementation. Here describe the implementation decisions, the trade-offs. Also, discuss the limitations, especially with regard to testing. Explain how you tested for security vulnerability and what is no tested.
- performance testing report
- team member contributions: Describe how the team worked, how often you met and how you collaborated. Then for each team member, 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.

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)
- readme file explaining how to run it; also how the TA should ran the test cases.

The **test war** file should contain
- client samples to test the web services
    - the clients are Java programs that send requests (REST), receive the replies as
- scalability client (see slides), that sends requests to the application.

GRADING:
80% on meeting the specs (both system functionality and report contents)
20% on internal quality (design, style, code, etc.), robustness (exemplified by test cases)

SQL schema for bookstore database (this was tested on MySQL)

-use it as a starting example
-you might need to fix it for other databases
-you can extend it as you see fit.

```
/** bid:     unique identifier of Book (like ISBN)
* title:    title of Book
* price:    unit price WHEN ordered
* author:   name of authors
* category: as specified
*/

DROP TABLE  if exists Book;

CREATE TABLE Book (
 bid      VARCHAR(20) NOT NULL,
 title    VARCHAR(60) NOT NULL,
 price    INT NOT NULL,
 category  ENUM('Science','Fiction','Engineering') NOT NULL,
 PRIMARY KEY(bid)
);
```

```
#
# Adding data for table 'Book'
#

INSERT INTO Book (bid, title, price, category) VALUES ('b001', 'Little Prince', 20, 'Fiction');
INSERT INTO Book (bid, title, price, category) VALUES ('b002','Physics', 201, 'Science');
INSERT INTO Book (bid, title, price, category) VALUES ('b003','Mechanics' ,100,'Engineering');

#



/* Address
 * id:      address id
 *
 */
DROP TABLE if exists Address;

CREATE TABLE Address (
 id              INT   UNSIGNED NOT NULL AUTO_INCREMENT,
 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)
);

#
# Inserting data for table 'address'
#

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');
#
#



/* Purchase Order
 * lname:           last name
 * fname:           first name
 * id:              purchase order id
 * status:status of purchase
 */

DROP TABLE if exists PO;


CREATE TABLE PO (
 id       INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
 lname     VARCHAR(20) NOT NULL,
 fname     VARCHAR(20) NOT NULL,
 status    ENUM('ORDERED','PROCESSED','DENIED') NOT NULL,
 address   INT UNSIGNED NOT NULL,
 PRIMARY KEY(id),
 INDEX (address),
 FOREIGN KEY (address) REFERENCES Address (id) ON DELETE CASCADE
);

#
# 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');
#
#



/* Items on order
 * id :       purchase order id
 * bid:  unique identifier of Book
 * price: unit price
 */

DROP TABLE if exists POItem;

CREATE TABLE POItem (
 id       INT UNSIGNED NOT NULL,
 bid      VARCHAR(20) NOT NULL,
 price    INT UNSIGNED NOT NULL,
 PRIMARY KEY(id,bid),
 INDEX (id),
 FOREIGN KEY(id) REFERENCES PO(id) ON DELETE CASCADE,
 INDEX (bid),
 FOREIGN KEY(bid) REFERENCES Book(bid) ON DELETE CASCADE
);

#
# Inserting data for table 'POitem'
#
INSERT INTO POItem (id, bid, price) VALUES (1, 'b001',  '20');
INSERT INTO POItem (id, bid, price) VALUES (2, 'b002',  '201');
INSERT INTO POItem (id, bid, price) VALUES (3, 'b003',  '100');
#
#

/* visit to website
 * day:                date
 * bid:    unique identifier of Book
 * eventtype:        status of purchase
 */

Inserting TABLE if exists VisitEvent;
```

```
CREATE TABLE VisitEvent (
 day                varchar(8) NOT NULL,
 bid        varchar(20) not null REFERENCES Book.bid,
 eventtype          ENUM('VIEW','CART','PURCHASE') NOT NULL,
 FOREIGN KEY(bid) REFERENCES Book(bid)
);

#
# Dumping data for table 'VisitEvent'
#
INSERT INTO VisitEvent (day, bid, eventtype) VALUES ('12202015', 'b001', 'VIEW');
INSERT INTO VisitEvent (day, bid, eventtype) VALUES ('12242015', 'b001', 'CART');
INSERT INTO VisitEvent (day, bid, eventtype) VALUES ('12252015', 'b001', 'PURCHASE');

#
#
```