CMPE 273: Enterprise Distributed Systems

Class Project (Spring 2020) - Simulation of Amazon.com

Team size: **5 to 6 people per team**
Due date: **May 8th, 2020**

1. **API Documentation:** **March 29th, 2020**
    - Submit your API report document on Canvas before 11:59 PM.
    - You must turn in a document describing the interface of your backend server consisting of all the API request- response descriptions.

2. **Presentation:** **May 1st, 2020**
    - Submit your presentation document on Canvas before 2:00 PM
    - Group wise presentation should be given in class meeting.
    - Describe the system architecture used in your project.

3. **Project Demo:** **May 8th, 2020**
    - Submit the project report before 2:00 PM.
    - Demo of your application in class meeting.

4. **Peer Review:** **May 8th, 2020**
    - Submit the peer review document on Canvas before 11.59 PM.
    - Grade your team members except yourself on the scale of 10 and describe their contributions to the project.
    - This will be an individual submission and should keep the grades and comments confidential.

Amazon.com

Amazon.com, Inc., is the largest Internet-based retailer in the United States. It is an e-commerce web application where customers can search for products and place orders which get home delivered. It also allows product sellers to register and present their products for sale. The customers can add their choice of products into a cart and place the order. They can also keep track of their orders and see its status.

Project Overview:

In this project, you will design a 3-tier distributed application that will implement the functions of Amazon website. You will use the techniques used in your lab assignments to create the system. In Amazon prototype, you will manage and implement different types of users.

Though Amazon.com has wide range of features, this project is limited to only few important functionalities which cover all the essential features required to place an order and implement the functionalities which involve different roles.

For each type of object, you will also need to implement an associated database schema that will be responsible for representing how a given object should be stored in a database.
Your system should be built upon some type of distributed architecture. You have to use message queues as the middleware technology. You will be given a great deal of "artistic liberty" in how you choose to implement the system.

Your system must support the following types of entities:

a. Customers
b. Sellers
c. Admin users

Your project will consist of three tiers:
- The client tier, where the user will interact with your system
- The middle tier/middleware/messaging system, where the majority of the processing takes place
- The third tier, comprising a database to store the state of your entity objects

Tier 1 – Client Requirements

The client should have a pleasing look and be simple to understand. Error conditions and feedback to the user should be displayed in a status area on the user interface. Try to create a UI similar to Amazon.

Your application should not allow customers to search for products and look at product details without login as Amazon does. They should be able to view the search page only after successful login.

Login & Sign up

- User should be able to create a new account by providing his Name, email id and password.
- A user can sign up either as a Customer or a Seller.
- In case of Seller, the name should be treated unique.
- User should be able to login using his email id and password.
- There should be single login page for customer, seller and admin.
- You can create credentials for admin from backend.
- Based on their account type, they should be redirected to their respective pages.

Product Search Results page

- Customer should be able to search any product based on product name or seller name.
- Customer can also apply filter on the category of products before searching.
- Customer can sort the results based on price or ratings in both ascending and descending.
- Customer should be able to apply filter the results based on ratings and price.
- Pagination should be used to display search results. (Server side recommended for better performance).

Product Details page

- Every product page should the following details:
    - Product name
    - Seller name
    - Ratings (Stars)
    - Price
    - Description
    - Multiple images (similar to Amazon)
    - Offers/Promotions/Discounts (optional)
    - Customer ratings and reviews
- Clicking on Seller name should take the user to Seller profile page
- Customer should be able to choose a quantity and add the product to cart.
- Customer should be able to view all the images of the product one by one.
- Customer should be able to add a comment and also add rating to the product.
- Customer should be able to view the comments added by other customers.

Customer Profile & Activity

- User should be able to view his profile, update his name and profile picture.
- User should be able to view the counts of votes he has added, comments he has provided.
- User should be able to view the list of comments he has added on different products.
- Clicking on each product comment should take the user to the product page.

Seller Profile Page

- When a customer clicks on any Seller's name, it should take him to Seller's profile.
- In this page, the basic details of seller like Name, Address and Profile Image should be displayed along with all the products available under that Seller.

Saved Addresses

- User should be able to add any number of addresses to his saved addresses directory which he can choose one from during Check out while placing an order.
- User can also update and delete any saved address added by him previously.
- Each address should comprise of the following fields:
    - Full Name
    - Street Address line 1
    - Street Address line 2
    - City
    - State
    - Country
    - Zip Code
    - Phone Number

Saved Payment Options

- User should be able to add a payment method by entering their credit/debit card details and he can choose one of them during Check out while placing the order.
- User can also edit or delete any card added by him previously.
- Each payment method should comprise of the following fields with proper validations:
    - Name on Card
    - Card Number
    - Expiration Date
    - Security code or CVV

Cart

- In Cart page, user should be able to review all the items he wants to place an order.
- The products added to cart should be displayed along with their seller name, quantities individual price and total price.
- User should be able to change quantity or remove product from the cart.
- User can move the product from Cart to Save for later.
- Users can choose if the product is a gift and enter a gift message only if it is gift.
- Increase the price of the product slightly if the product is a gift
- Allow user to check out the products in the Cart.

Save for later

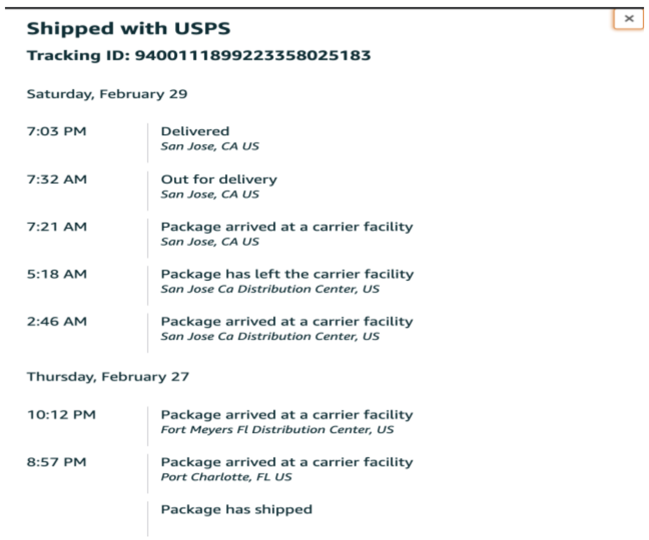
- Save for later is a list of products which user likes and would like to purchase later.
- User should be able to delete items from the Save for later.
- User should be able to move the products to Cart.

Checkout

- Customer should be able to pick a saved address or enter a new address for delivery.
- You need not add the newly added address to saved addresses.
- Customer should be able to pick a saved card or add a new card for payment.
- Newly added card should be added to saved cards if it does not already exist in the list.
- Only after entering Address and Card details, the customer should be able to place the order.
- You are free to provide any discounts or add delivery charges and taxes in the final bill.
- After the order is placed, the customer should see the order summary page with the status of the order.

Orders

- List of all the orders placed, their order status, order details must be displayed.
- On clicking on each order, the customer should be able to see the billing details, payment details, delivery address.
- On clicking on the product in the order, the customer should be able to reach the product page.
- Customer should be able to view the list of all the updates for each product under each order.



Shipped with USPS
Tracking ID: 9400111899223358025183

Saturday, February 29

7:03 PM Delivered
San Jose, CA US

7:32 AM Out for delivery
San Jose, CA US

7:21 AM Package arrived at a carrier facility
San Jose, CA US

5:18 AM Package has left the carrier facility
San Jose Ca Distribution Center, US

2:46 AM Package arrived at a carrier facility
San Jose Ca Distribution Center, US

Thursday, February 27

10:12 PM Package arrived at a carrier facility
Fort Meyers Fl Distribution Center, US

8:57 PM Package arrived at a carrier facility
Port Charlotte, FL US

Package has shipped

- Customer can cancel individual products in any order as long as the product status is not changed to delivered.
- You need not implement returning of products after delivery.
- There should be a separate tab which display Open orders (Not delivered yet) with an option for user to Cancel the order.
- There should be a separate tab which displays Cancelled orders

Seller

A seller will be having a different set of pages. He will not be able to view Customer's pages. Please refer any references related to **Amazon Seller Central.**

Seller Profile page

- A seller should be able to view, edit and update his profile page containing his name, address and profile image.

Manage Inventory page

- In this page, a seller should be able to view all the products added by him.
- Pagination should be used to display all products.
- Seller should be able to search the products added by him.
- A seller should be able to add new products, update existing products and remove products.
- You are supposed to handle the scenarios where the product is already ordered by the customers and when the seller removes the product, customer and seller should still be able to see the product in the orders. A removed product should not be appearing in search results.
- Pagination should be used to display products.

Add Product page

- The seller should add all the required details of a product:
  - Product name
  - Price
  - Product category (e.g. Electronics, Dressing, Cosmetics etc) – dropdown
  - Description
  - Multiple images (up to 5)
  - Offers/Promotions/Discounts (optional)

Orders page

- List of all the orders placed which contains product sold by the seller with recent order on top.
- On clicking on each order, the seller should be able to see the billing details, payment details, delivery address.
- The order should show details of the products which are sold by the seller only.
- Seller should be able to view the list of all the updates in the order.
- Seller can cancel the order or change the status to Packing or Out for Shipping.
- There should be a separate tab which display Open orders (Not delivered yet) with an option for user to Cancel the order.
- There should be a separate tab which displays Delivered orders and Cancelled Orders (cancelled by customer or seller).

Reports

- Seller should be able to see a statistics of all the products sold by him, their quantities and the amount earned.
- Seller should also be able to see the monthly amount earned by sales of his products.

Admin User

They are the service staff working for Amazon who will have to keep a track of all the sellers and orders in the website. They will be able to view different set of pages.

Inventory Listings

- They should be able to add or remove the product categories to the application. These categories should be available in the dropdown for the Seller while adding their product and for Customers while searching products.
- They should not be able to remove the categories which have products mapped in them.
- When they click on each category, they should be able to view all the products added under that category with minimum details about the product – seller, price etc.

Seller Profile Page

- Admin users should be able to see the list of sellers in the system.
- They should be able to search for sellers using Seller name.
- On clicking a Seller name from results, they should be able to view all the products under the seller.
- Admin user should be able to view month wise sales amount for that seller.

Orders Page

- Admin user should be able to view a list of all the orders placed in the application and should be able to filter by seller name (by search text) or by selecting order status.
- Admin user should be able to move the status from Out for Shipping to Package arrived, Out for Delivery and Delivered.

Analytics Dashboard

- Admin user should be able to view different types of analytics graphs showing following data:
    - No of orders per day.
    - Top 5 most sold products.
    - Top 5 sellers based on total sales amount.
    - Top 5 customers based on total purchase amount.
    - Top 10 products based on rating.
    - Top 10 products viewed per day.

Tier 2 — Middleware

You will need to provide/design/enhance a middleware that can accomplish the above requirements. You have to implement it using REST based Web Services as Middleware technology. Next, decide on the Interface your service would be exposing. You should ensure that you appropriately handle error conditions/exceptions/failure states and return meaningful information to your caller. Your project should also include a model class that uses standard modules to access your database.

Use Kafka as a messaging platform for communication between front-end channels with backend systems.

Tier 3 — Database Schema

- You will need to design your database to store your data in both relational and NoSQL databases. Choose column names and types that are appropriate for the type of data you are using.
- You will need to decide which data will be stored in MongoDB and MySQL. (Hint: Choose the database for every information stored based on pros and cons of MongoDB vs MySQL)
- You need to implement SQL caching using **Redis**

Scalability, Performance and Reliability:

The hallmark of any good object container is scalability. A highly scalable system will not experience degraded performance or excessive resource consumption when managing large numbers of objects, nor when processing large numbers of simultaneous requests. You need to make sure that your system can handle many users and incoming requests. Pay careful attention to how you manage "expensive" resources like database connections.

Your system should easily be able to manage 10,000 users and 10,000 products. Consider these numbers as **minimum** requirements. Further, for all operations defined above, you need to ensure that if a particular operation fails, your system is left in a consistent state. Consider this requirement carefully as you design your project as some operations may involve multiple database operations. You may need to make use of transactions to model these operations and roll back the database in case of failure.

Testing:

To test the robustness of your system, you need to design test cases to test all the functionalities that a regular client would use. You can use your test cases to evaluate the scalability of the system as well by running the tests with thousands of products and users. Use these tests to debug problems in the server implementation.

- **Mocha Testing:** Implement testing on **Ten** randomly selected REST web service API calls using Mocha and include the results in the project report.

Presentation:

1. Group number and team details
2. Database Schema
3. System Architecture Design Diagram
4. Performance comparison bar graphs that show difference in performance by adding different distributed features sat a time such as B (Base), S (SQL Caching), D (DB connection pooling), K (Kafka) for 100, 200, 300, 400 and 500 simultaneous user threads. (5 Bar graphs in total).
   Combinations are:
   a. B
   b. B+D
   c. B+D+S
   d. B+D+S+K
   e. B+D+S+K+ other techniques you used

   Note: Populate DB with at least 10,000 random products and measure the performance on an API fetching the products.
5. Performance Comparison of services with below deployment configurations with Load balancer
   a. 1 Services server
   b. 3 Services server
   c. 4 Services server

Project Report:

Your project report must consist of the following:

- A title page listing the members of your group
- A page describing how each member of the group contributed for the project. (One short paragraph per person)
- A screen capture of your database schema
- System Architecture Design diagram
- A short note describing:
  - Your object management policy
  - How you handle heavy weight resources
  - The policy you used to decide when to write data into database
- Screen captures of your application (few important pages)
- A code listing your server implementation for entity objects
- A code listing of your server implementation of the security and session objects
- A code listing of your main server code
- A code listing of your database access or connection
- A code listing of your Mocha test and output result screenshots
- Observations on performance of the application based on the comparative analysis at different deployment configurations with graph

Scoring Points:

- **Functionality** (40 marks) – Every functionality mentioned in the requirement above carry some marks and they will be randomly tested along with the code and ESLint.
- **Scalability** (15 marks) – Your project will be tested with thousands of objects. There should not be any degradation in performance of application and it should not crash for any reason.
- **Distributed Services** (15 marks) – Divide the client requirements into different services and deploy them on the cloud. Each service should be running on the backend server connected using Kafka. Divide the data into MongoDB and MySQL and provide reasons for the choice of database for every object.
- **UI** (15 marks) **–** It is important that the user interface of your application is similar to Amazon.com.
- **Report** (15 marks) **–** Performance testing results, Mocha testing results and your explanations in Project Report carry marks.

Notes:

1. Maintain a pool of database connections. Opening a database connection is a resource-intensive task. It is advisable to pre-connect several database connections and make use of them throughout your database access class so you don't continually connect/disconnect when accessing your database.
2. To prevent a costly database read operation, you can cache the state of the entity objects in a local in-memory cache. If a request is made to retrieve the state of an object whose state is in cache, you can reconstitute the object without reading the database. Be careful to update the cache, when the state of the object changes. You are required to implement **SQL caching using Redis**.
3. Don't write data to database that didn't change since it was last read.
4. Focus on the implementation of all the features first. Complete implementation that passes all feature tests is worth as much as performance and scalability part of the project.
5. Do not over optimize. Groups in past in attempt to optimize the project, implemented complex optimizations but failed to complete the feature set.
6. Try to implement all the features with proper exception handling and user-friendly errors. Implement proper input validations in all pages. Do not give scope for the user to make the application crash.
7. Try to implement all the features as similar as it is in Amazon.com.

Other Notes:

Implement following restrictions on input fields

- *State abbreviation parameters*

State abbreviations should be limited to valid US state abbreviations or full state names. A Google search will yield these valid values. Methods accepting state abbreviations as parameters are required to raise the exception if the parameter does not match one of the accepted parameters. You do not need to handle US territories such as the Virgin Islands, Puerto Rico, Guam, etc.

- *Zip code parameters*
  - Zip codes should adhere to the following pattern:
    [0-9][0-9][0-9][0-9][0-9] or [0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]
  - Examples of invalid Zip codes:
    1247, 1829A, 37849-392, 2374-2384