institute for SOFTWARE RESEARCH

Engineering Data Intensive Scalable Systems 17-648

Project 4

Objectives:

This project is designed to give you hands on experience with various aspects of data intensive distributed systems (the scalable part comes later). Much of the complexity of both dealing with data and the distribution is often abstracted away from the application developer. The issue is that as systems grow in scale these abstraction mechanisms start to break down. As the systems scale the burden on the application developer increases.

The individual projects are designed to force you to deal with these complexities by removing the abstractions prior to building a system at scale. Specifically this assignment is designed to:

- Gain experience with availability patterns and tactics
- Understand how these tactics and patterns impact other quality attributes
- Reason about consistency in a distributed environment
- Recognize how state management is related to systemic properties of interest
- Prepare the system to support scalability

Background:

In addition to all of the capability that you had in your previous system you have now begun to experience the pain of having your system go down. Recognizing that you system is built from faulty components you need to add in some ability to withstand faults.

Project:

Based on an analysis of your system you've determined the kinds of faults that you need to be able to handle. They include:

- A failure of your web server: If the web server has an issue this should be transparent to the user. They should still be able to connect to the system using a browser (you might want to see http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html).
- A connectivity issue between the ordering system and one or more warehouses. The system should still be able to allow the user to place an order and should resolve any inconsistency when connectivity is restored. If it turns out the bike is not actually available the system should record the fact that the bike is backordered and not bill the customer until the bike becomes available.
- Failure of the warehouse system: If some part of the warehouse system goes down the system should still be able to access the warehouse data, query inventory, and place orders.

In addition to figuring out how to handle these issues you need to think about data consistency. What is your consistency model going to be? If you decide that you'll implement "eventual consistency" you need to think about how long the system could be inconsistent. What impact does your decisions have on how long it takes to become consistent again?

You also need to think about how your system could scale. You don't need to be able to add nodes to support scalability but you do want to understand how your decisions do or do not support scalability.

The primary objective for your company is still to be able to sell as many bikes as possible. When making design decisions consider the impact of the decision on the ability of the system to facilitate the sale of bikes.

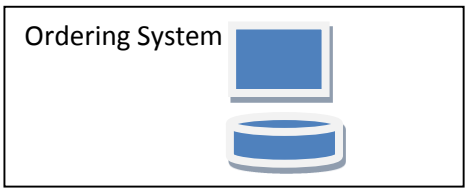
The project will continue to support a web interface for the ordering system. The previous requirements remain. The system should work correctly with multiple users. This means that you are going to have to figure out a way to manage multiple users accessing the system at the same time and ensure the data is correct and consistent.

The functional requirements for the system are:

- Create an account that includes:
  - A first name
  - Last name
  - Address
  - a username and password
- Browse inventory
  - Order inventory by
    - Price
    - Bike name

- Purchase bikes (must be logged in). **The customer can buy more than one bike (potentially different models).** For each order you should store (the customer information can be gotten from the account):
    - Customer first and last name
    - Shipping address
        - Address
        - City
        - State
        - Zipcode
    - Items purchased
        - Item number(s)
        - Item name(s)
        - Price(s)
        - Total price
        - Quantity
- Search order history (must be logged in)
- Send notification that the bike has shipped
    - The warehouse should indicate when the bike has shipped (this can be some random timeout after the order is placed)
    - The system should send a notice to the user (we can assume that this would be an email but in this case it can just be a pop up notification)

You will continue to have access to the data sets for the two warehouses. The warehouses have separated the inventory data (the current bikes in stock) from the catalog data (the details of the bikes that the supplier typically stocks). You should make sure that the inventory across the system is as consistent as possible.



You are constrained to using Java SE 7 or later and Apache Tomcat 7 or later. You can use javascript, JSP, or Servlets for the processing of data on the web browser. Chrome will be the target browser.

Deliverables:

You should deliver your source code and execution instructions. The code and instructions should be zipped together and submitted via blackboard by the due date. Late assignments will be penalized one letter grade per day late.

The instructions should describe exactly what we need to do in order to compile and execute your software. If we are not able to execute your software based on the instructions provided points will be deducted. You will not be given an opportunity to fix issues (so be sure to test your software and instructions on a new machine to make sure no assumptions are made on the configuration of the environment).

Evaluation:

You have a wide range of latitude with respect to how you implement this system. You will be evaluated along several dimensions:

- Quality of your implementation: It's expected that you adhere to acceptable coding standards. The more you use generally acceptable practices as opposed to just "hacking" the solution together the better you will do along this dimension. It's also expected that the system works. A fully working system is better than a partially working system, a partially working system is better than a system that builds but doesn't execute, and so forth.
- Understanding: You'll be evaluated based on your level of understanding with respect to what you've implemented and the impacts of these solutions. These will be discussed orally after you submit your assignment.