Project Overview

Client Background

Shoppers are a well-established mass-merchandise retailer with operations in Australia, Canada and India. They have hundreds of stores scattered throughout each of these geographies and they cater to a diverse customer base, selling thousands of items from each store.

Business Problem

Shoppers.com has functionality to allow customers to browse, shop, find store locations etc. Recently they started facing scalability issues, leading to lost business and customer dissatisfaction.

Colin Hall, the CEO, has defined this as the company’s highest priority, as he believes there is significant revenue loss especially around festival times like Christmas, and they are currently losing business to other retailers having more robust and scalable websites.

Colin believes that this initiative will have a dramatic impact on the customers’ perception of the company and he is not prepared to take any half measures. He wants to solve this situation on a long term basis.

Business Requirements

* They don’t want to solve the scalability issue by ‘scaling up’ the complete application and investing in infrastructure. This is cost prohibitive, also not a good long term solution.
* The scalability issues being faced are not for the complete application, but only for specific functionalities around product catalogue management.
* The organization had embraced SOA as a de-facto standard and is ready to invest for moving towards a Service-Oriented approach.
* Besides scalability Shoppers CEO is also interested in flexibility to evolve individual business areas independently.

Proposed Architecture

This section explains the proposed high level solution. You are free to expand the same or suggest better ways.

#### Microservices

Raymond, the architect decided to go with Microservice architecture. This would allow separation of the monolith backend logic to independently managed and hosted micro-services. These micro-services could also collaborate with each other.

He proposed to decompose the backend monolith API on the functional areas following the Single Responsibility Principle. Some of the initially identified functionalities include product catalogue management, price management, order management, inventory management, etc.

The proposed solution would allow each microservice to be implemented, managed and evolve independently. The separate deployment solves the scalability issues by ‘scaling up’ or ‘scaling out’ only the required functional area. Each service could be implemented using the technology of choice thus solving the vendor lock-in issue, providing the flexibility and reducing development complexity.

Required Deliverables

The following list of deliverables is required. It is important that you are able to have a working solution that you can demonstrate through automated test cases or any other tool of your choice.

Note - Please draw a logical diagram of this setup on the white board/ paper that include the services, the database, clients of the service. Please consider components that would help fulfil non-functional requirements such as Service discovery, load balancing, monitoring etc.

Once you are done with the high level architecture, you can start coding:

#### Product Catalogue Service

API that provides the functionality to

* Add a product
* Retrieve the list of products based on simple search criteria e.g. product type
* Remove a product from the catalogue.

The service should be independently hosted and accessible over http port. Any database of your choice or SQL Server could be used.

#### Pricing Service

API that provides the functionality to return the price for a given product

You need to interact with ‘Product Catalogue Service’ to retrieve any required details for the given product. The service should be independently hosted and accessible over http port. Any database of your choice or SQL Server could be used.

Solution Specifics

* The protocol of choice is HTTP/REST
* JSON is to be used as data interchange format
* Services to be implemented using technology of your choice and justify that technology.
* Any in-memory database / No SQL / RDBMS DB could be used.
* Separate database instances for each service. Assume no sync up required between these services.
* Services to be independently hosted
* Must create concrete application structure
* Must create structure of classes and interfaces required
* Must be able show the implementation of a design pattern
* Must implement at least one interface
* Must write one method
* Must write unit test using any mocking framework
* Must create a DB, sample tables and stored procedure
* Must create a service/API
* Must create a simple web page or MVC application to consume the service
* Should use AJAX, Quaere, Dependency Injection, ORM
* The requests/response in JSON formats.

**Assumptions and Risks**

**List down your assumptions and risks in building your solution**