Sigma – Shoppers.com

## Context

This document is intended to provide very high level draft design of Shoppers online shopping functionality and brief overview of solutions built.

## Logical Design Diagram

Below diagram shows various services and how they interact.



### Key Points

* Each service is responsible for well-defined individual functionality exposing REST API end point to communicate.
* Service Discovery - Each Service would register itself in Service Registry with location details. On request load balancer with reverse proxy/API Gateway would request Service Registry and gets service location details and then makes the call to service. Service registry will also poll each individual services for health check at regular intervals. Any crashes or failures it finds will lead to de-register the service as well send notification to monitoring service for alerting.
* Load Balancer (Reverse Proxy) – Each service would be hosted in multiple nodes to balance the traffic and scale the results. Load balancers with reverse proxy will have functionalities as
  + Discovering the service location by calling Service Registry
  + Load Balancing the traffic among nodes
  + Authenticate/Authorize the requests
  + Caching static data
* Logging & Monitoring – A separate internal service is responsible for managing the logs. Keeping logging service isolated reduces the load on main services. Individual services notifies the log details or incidents on asynchronous mode to Log API. Log & Monitor service also responsible for alerting on crashes or reaching the threshold.
* As given online portal is accessed in different regions, to scale the request and responses services will be hosted in Asia and American region data centers. Reverse proxy will redirect the request specific data center based user origin.

Note that this diagram doesn’t cover all services required for Shoppers but majority of functionality would repeat for other services.

## Logical Design of Individual Service

Below diagram depicts component design of individual service.



Each individual service consists of its own components as well shared components along with dedicated database.

* Data Access Layer – This is common component provides functionality to connect database and execute the stored procedure or query. This component also translates data into data model object.
* Infrastructure – This component carries common functionality across services and it’s an independent component.
* Logging and Monitoring – This component interfaces Logging service to notify logs and alerts. It’s a common component used across the services.
* Data Models – This component consists of entities structures specific to the service.
* Services – This component interacts with DAL to connect to DB specific to service and also provides processed data objects to controller.
* API Controller – This component is responsible for validating request and process the request by interacting with service components and send to user.
* Automated Tests – Responsible for unit tests of components fully covered
* Register Service – This component will notify Service Register about availability of service as location details to discover.

## Other Quality Items

1. Each service will be built independently and managed.
2. Services will be fully automated with unit tests and release builds. Build process would get source code from repository and execute tests and generate build.
3. Continuous integration and deployment is key to success of micro services solutions. Deployment process will be fully automated.
4. Circuit Breaker pattern could be applied to critical service calls like inventory and ordering for controlling resources and better monitoring.

## Risks

1. Load balancer and Service Registry are key components and plays very important roles in accessing the service. Availability of system fully depends on these two. Must make sure these two components are maintained with highly available infrastructure. Service Registry should have Disaster Recovery infrastructure for any emergency role over.
2. Synchronizing transaction among multiple services is difficult. In this case synchronizing inventory on placing order is key. Two phase commit approach would help.
3. Certain cases have to make multiple calls which might not be cost effective. In this case getting price for filtered product is costly process. This may require to rethink or brining price and produce into single service or making consolidated request to get price for all product. But overall might end up multiple service call to achieve single function due to distributed services. This is fully depend on effective design after understanding functionality.

## Assumption and Details of Shopper Solutions

This solution consist of two services Product Service and Price Service and a Web UI.

1. ASP.NET Web API being used to build REST API Services.
2. Dependency Injection being implemented in each service (Unity Application Block).
3. MS Unit and Moq are used for unit testing and mocking.
4. SQL Server 2012 Express edition is used for back end database.
5. Web UI is built using AngularUI libraries.
6. Jasmine and angular mocks are used for JavaScript unit testing.
7. Development environment
   1. VS 2015 (Open in administrative mode)
   2. IIS
   3. SQL Server
8. Once solution is built starting page URL

<http://localhost/ProductWeb/Product.html>

Note that this solution is very initial draft not fully implemented either functionality. This is just to showcase the features, patterns and technical concepts.