Solution Document for an E-Commerce Application/Platform

Version draft

Date : 20th Aug 2021

Author : Vijay Shankar Jha

Contents

Introduction

The purpose of this document is to describe solution for an e-commerce application/platform.

High Level Use Case Diagram

Figure 1 - Admin Use Case View

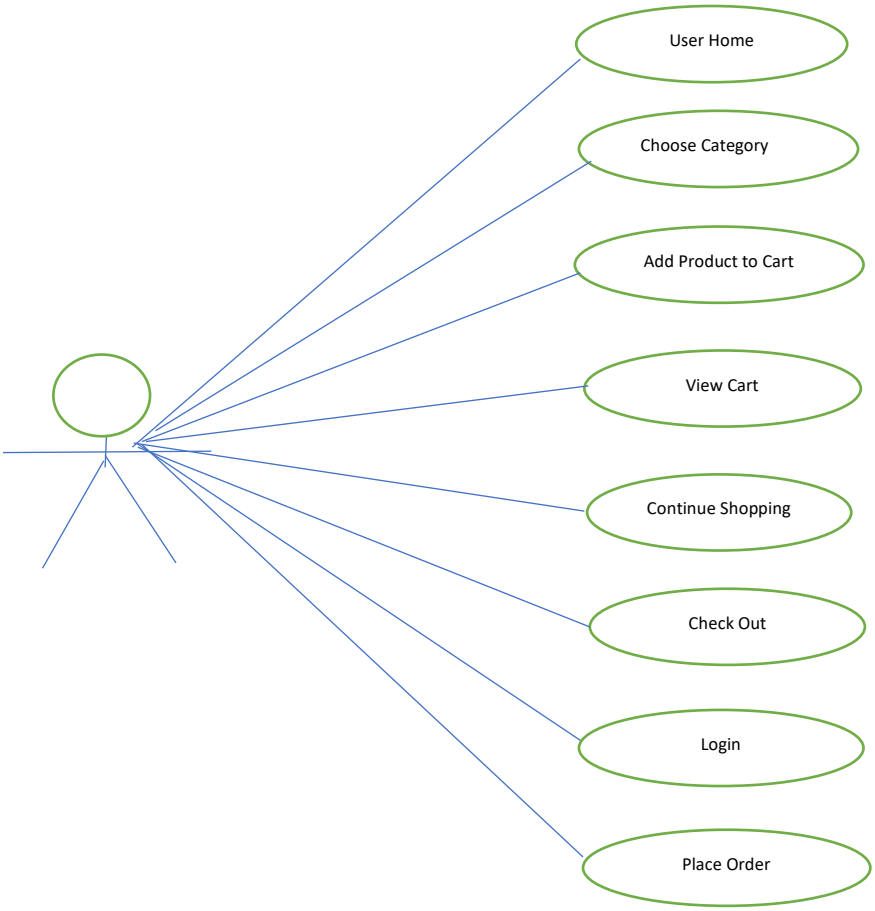

Figure 2 – User use case view

A Sample Use Case Description

Use-Case Number: UC-001

Application: e-commerce application

Use-Case Name: Login

Use-Case Description: It helps the User/Admin to check out items in the shopping cart by logging into the user-authentication form.

Primary Actor: User/Admin

Precondition: There is at least one item in the shopping cart to check out the items and to login to the user-authentication form.

Post-condition: The user is successfully able to log in.

Basic Flow:

- Run the application
- Go to the view-cart page
- Click the checkout button
- Enter the username and password
- Login/Register.

Exceptional Flow:

- Run the application
- Go to the view-cart page
- Click the checkout button
- Enter an incorrect username and password.
- Login/Register fails

Activity Diagrams

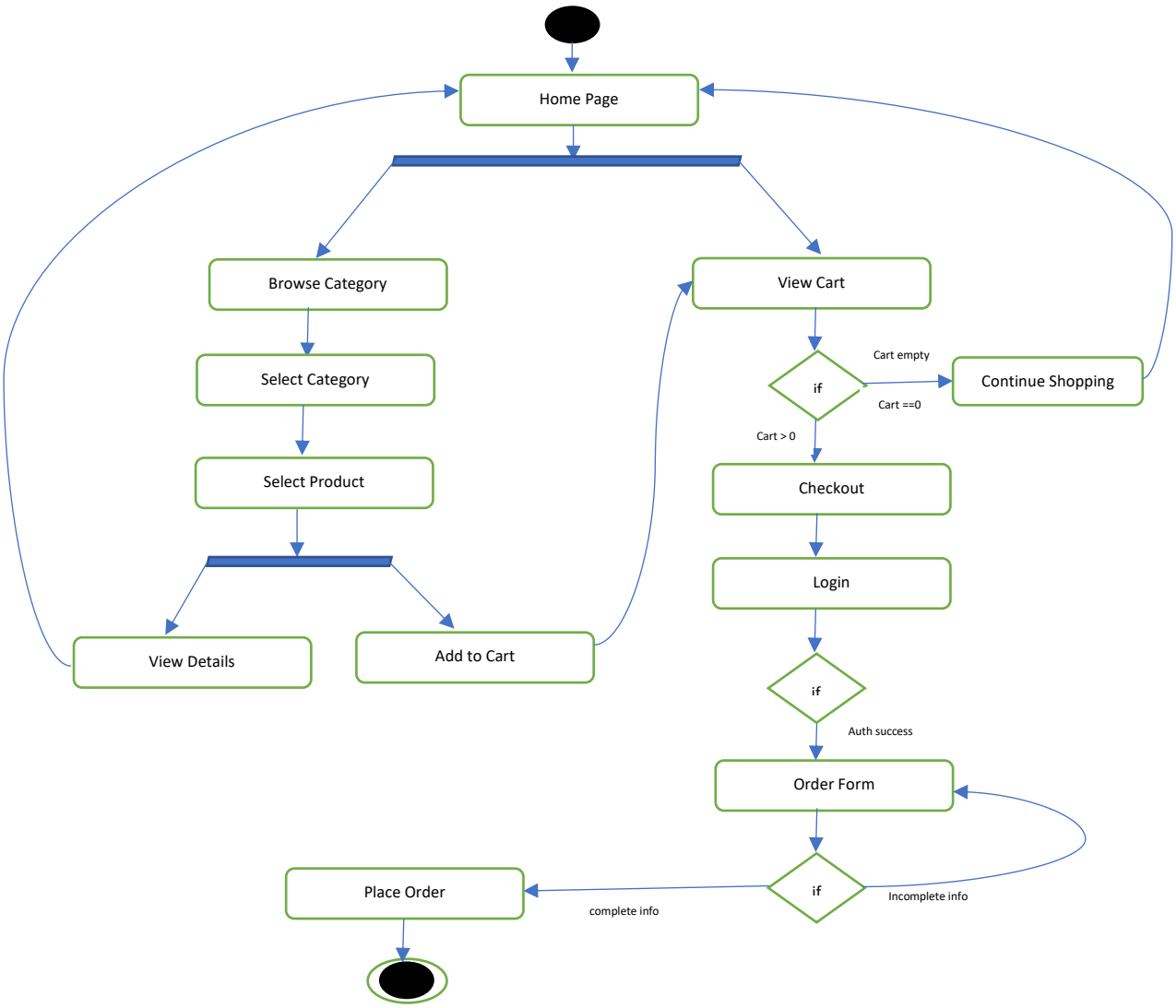

Figure 3 Shopping Activity diagram

Class Diagram

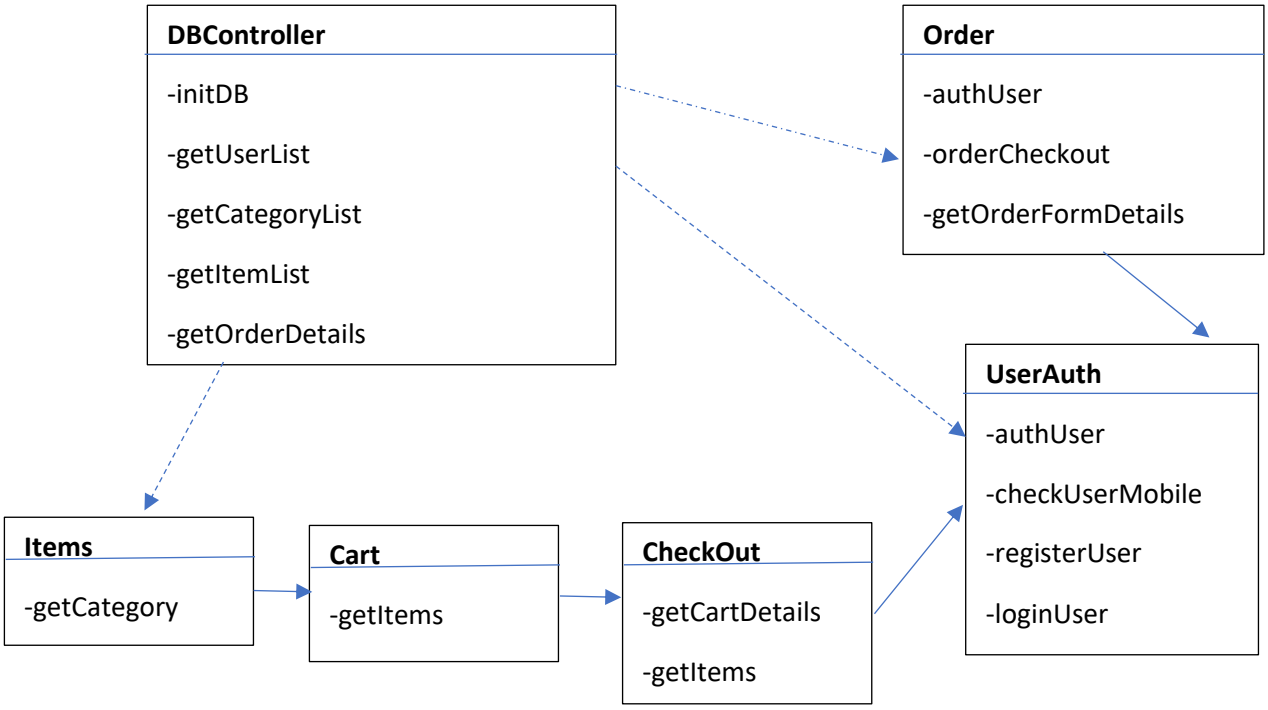

Figure 4 A Trimmed down class diagram

UI

To do

Test Cases

To do

Database Types

Hybrid Database type is to serve different purposes of the application such as structured and unstructured data and in-memory data.

Transaction Tracking—Relational Database or NoSQL Database

Product Catalog-- NoSQL Database document/key value store

Static Content—Object Store

In-Memory Data-- for in-memory database computing and performance optimization

Key-Value Store-- for caching.

Application Architecture

Microservices based architecture is more suitable for modern e-commerce kind of application and platform which provide following benefits:

Scalability -- How our services can be scaled on demand?

Availability -- How can we ensure that our services are available all the time or meet SLA?

Resiliency -- How our services can be made fault tolerant?

Independent, autonomous -- Are our services independent and autonomous?

Decentralized governance -- Can we manage services end to end in DevSecsOPs independently?

Failure isolation-- What happens if one service is not available and some composite service is also using it?

Auto-Provisioning -- Can a service be provisioned based on an event?

Continuous Integration and Delivery through DevSecOps -- Are services using CI/CD for DevSecOps?

Checkpoint Whether APIs are following 12-factor guidelines or at least implemented versioning and metering if APIs are exposed to public/partner?

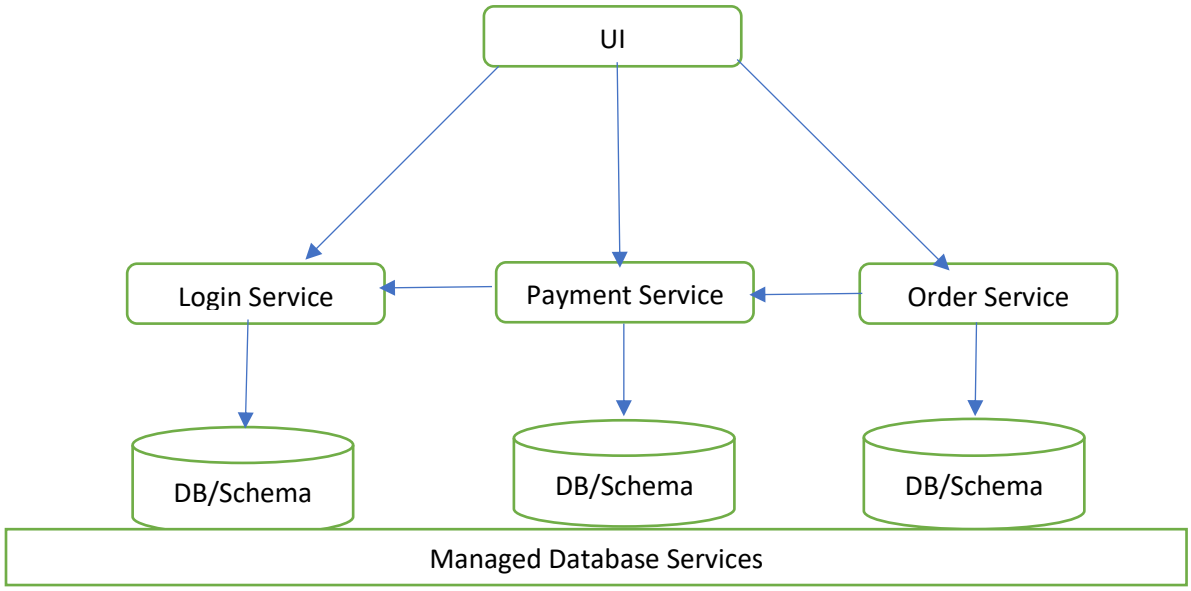

Figure 5 Microservice based application architecture

Frontend service is recommended for UI

Implemented Micro Services Design Patterns

1. API gateway pattern
2. Aggregator – order service
3. Asynchronous messaging – service inter communication messaging
4. Shared Data – Managed DB service
5. Event Sourcing
6. Command Query Responsibility Segregation CQRS – read from read replica and write to primary DB
7. Circuit Breaker (Order to Payment service)

Inter Service Communication using Queue and Topic using AWS MQ/SNS or RabbitMQ

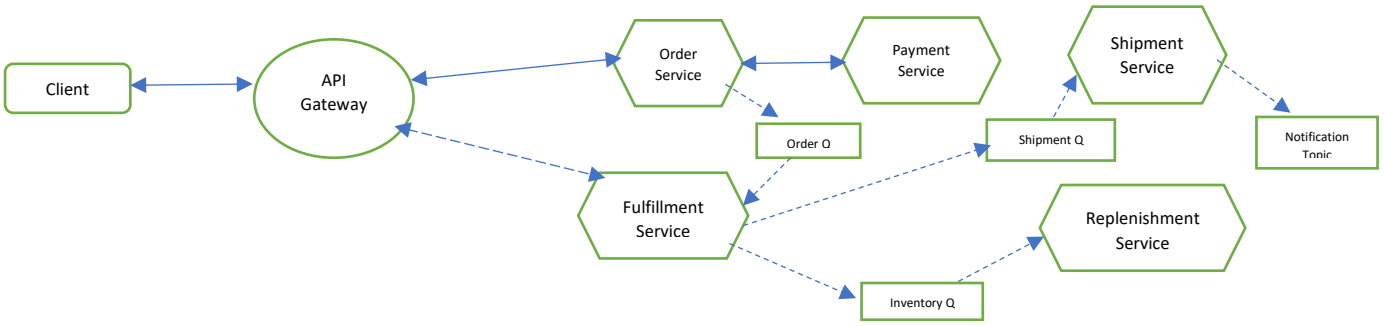

Figure 6 Order Service Flow

Deployment View

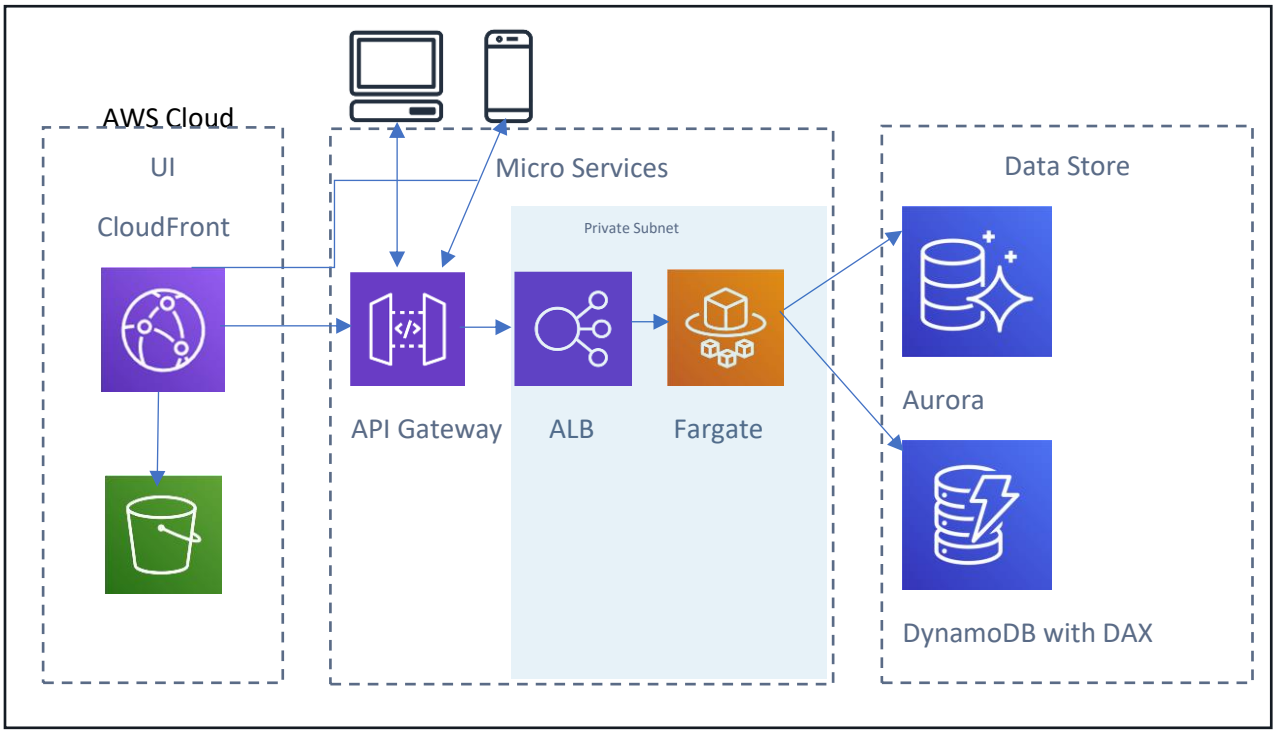

Figure 7 Serverless Architecture Using AWS Fargate

Order History View

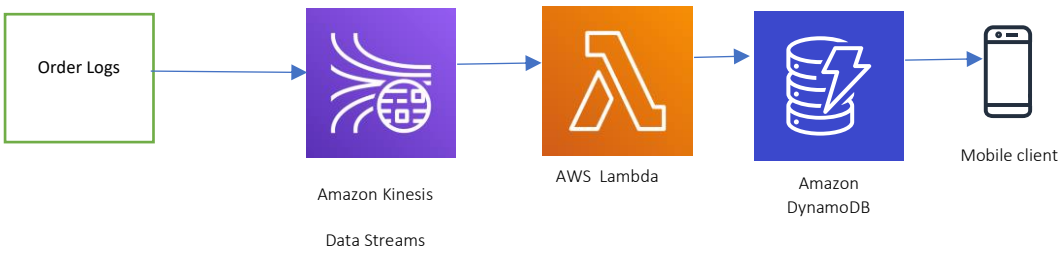

Figure 8 Order History

Product Recommendation

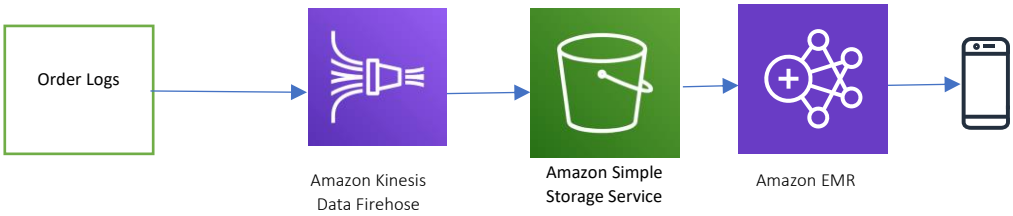

Figure 9 Product Recommendation

Near Real Time Log Analysis

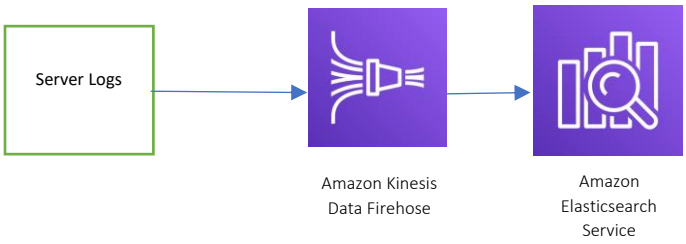

Figure 10 Near Real Time Log Analysis

NFRs

Scalability/Availability

1. Docker containers are used with AWS Fargate, so it's not necessary to care about the underlying infrastructure.

2. Amazon DynamoDB, Amazon Aurora Serverless is used, which is an on-demand, auto-scaling configuration for Amazon Aurora (MySQL-compatible edition), where the database will automatically start up, shut down, and scale capacity up or down based on application's needs
3. DynamoDB DAX with serve as caching layer without extra line of coding in application
4. Static contents are served from S3 which is scalable with virtually unlimited space

Performance

1. DynamoDB DAX with serve as caching
2. Multiple read replica of Aurora
3. Content distribution through CloudFront CDN
4. VPC endpoints to access AWS service like S3 and DynamoDB access
5. Caching at API and Database level

Cost

1. S3 bucket with domain name to serve content from S3 directly
2. Use SSL at CloudFront and API gateway level to reduce cost
3. Use S3 storage wisely
4. Private links for AWS service access and partner service access if they are on AWS e.g. payment service provider, supplier
5. Single API Gateway in the architecture across multiple web portal applications and microservices help achieve reusability of components and cost optimization
6. VPC endpoints to access AWS service like S3 and DynamoDB access

Security

1. Use public and private subnet
2. Only CloudFront and API Gateway with SSL accessible to public
3. Manage inter subnet communication through security groups and NACL
4. For public access open only port for secure access/https 443
5. Privatelinks and for partner service also if they are on AWS
6. VPC endpoints to access AWS service like S3 and DynamoDB access
7. Cloudfront, SSL, S3 with OAI (Origin Access Identity) and DDOS protection through AWS Shield
8. WAF on API gateway for OWASP top 10 and other cyber-attack protection

Miscellaneous

1. Service Discovery through API Gateway is preferred, but can be done with DNS-Based Service Discovery and Service Mesh also with API gateway is not used

2. As consequence of the CAP Theorem, distributed microservices architectures inherently trade off consistency for performance and need to embrace eventual consistency.
3. Step function-based service orchestration
4. AWS CloudWatch with S3 for centralized logging and monitoring
5. Distributed Tracing through X-Ray
6. Log analysis with Amazon Elasticsearch Service and Kibana
7. Log analysis with Amazon Redshift and Amazon QuickSight
8. Log analysis on Amazon S3
9. Protocols http/https
10. Message exchange through JSON
11. Auditing through AuditTrail and real time event analysis and actions
12. Resource Inventory and Change Management through AWS Config

--end of the document--