☐ Ecommerce Store

Create an **online store** where you can buy **video games** and **fan gear.**

Brief

Gone are the times that you need to find a local game store to get the latest Halo release. Instead, you can get any game you want, and some sweet gear, right on the internet.

Level 1

For someone to be able to purchase a game, they need to be able to find it on your website.

Create a website that includes pictures and names of some video games and fan gear.

Level 2

Managing inventory and the products in a store is a pain if you have to update code every time. We can use tools like content management systems to dynamically add products.

Integrate a CMS that allows you to manage available products.

Level 3

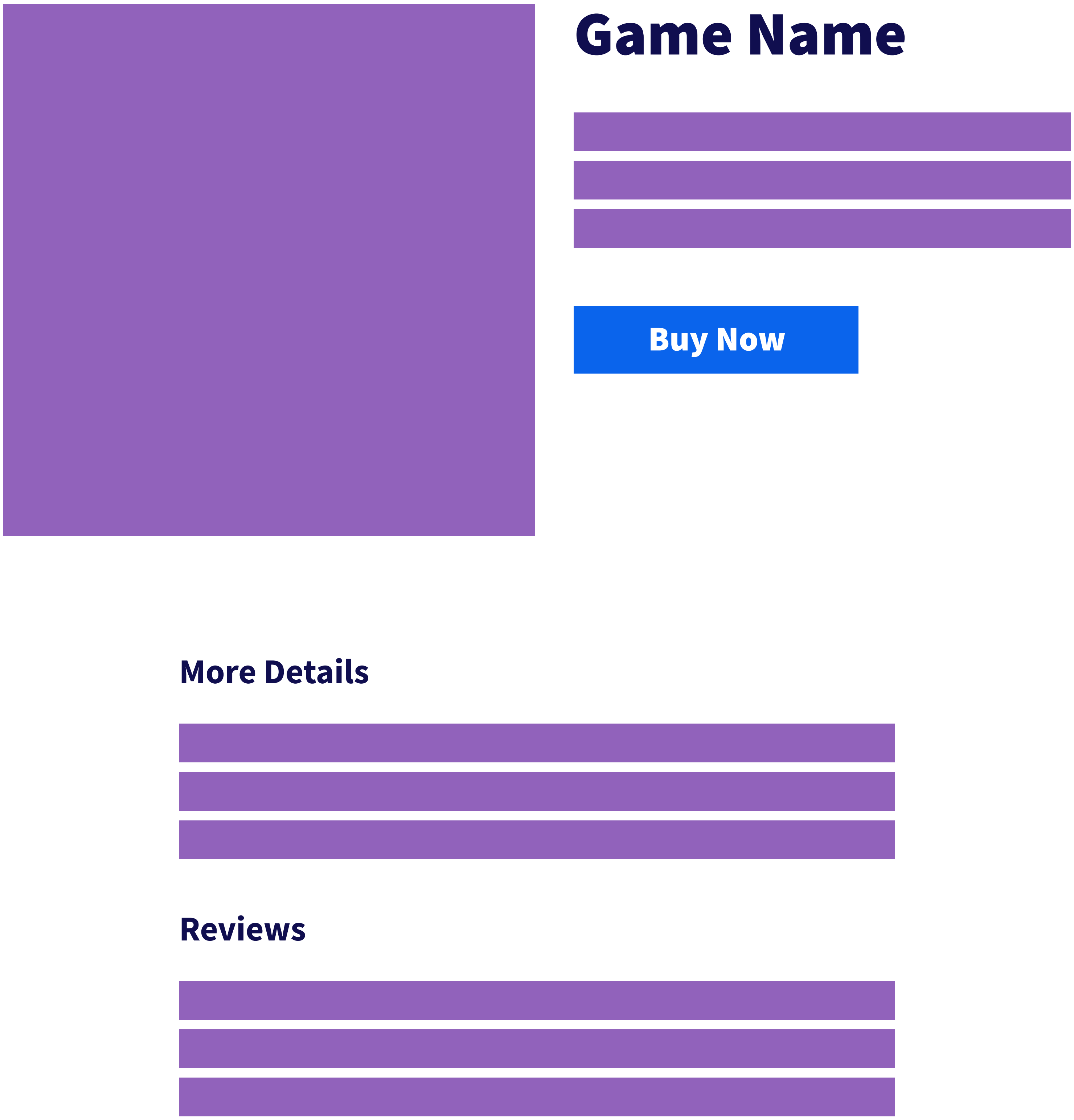
In order to purchase a product, we'll need to provide a way for the customer to pay for it. This includes adding the product to a cart and the checkout process.

Add a cart and payment provider that allows someone to purchase a game.

To Do

- ☐ Create a store
- ☐ Create a list of games
- ☐ Add games to website
- ☐ Create a list of fan gear
- ☐ Add fan gear to website
- ☐ Create CMS project
- ☐ Migrate content to CMS
- ☐ Source content from CMS
- ☐ Create shopping cart
- ☐ Set up payment provider
- ☐ Create checkout process

Layout Idea



Ecommerce Store – Backend Architecture

By Following the Domain Driven Design, we can break the monolithic application into number of small services, including:

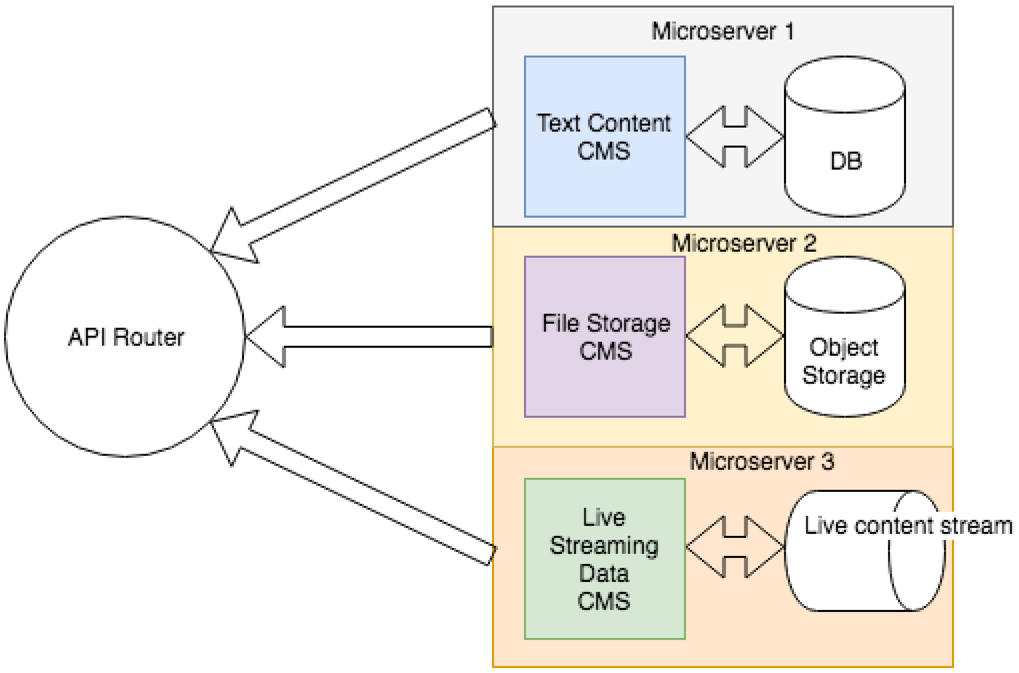
- An auth-service is serving the operations of sign in, sign up and sign out.
- Simple CMS Service to manage the products
- Product Catalog Service
- Cart Service
- Order Service
- Payment Service
- Config Service – Provides Configuration for all other services
- Service Registration Service
- An API Gateway which is just responsible for routing the incoming requests to downstream services.

The databases are also aligned to Microservice architecture, each of the service may have their own databases

CMS Microservice

The aim is to create small, flexible, and extendable content management system based pure on Spring Boot and MySQL or Mongo.

- It will expose REST APIs that will give access to products, related images, and documents.
- Client application will access by using API Gateway pattern



Method	Path	Description
GET	/{content-type}	Get a list of {content-type} entries
GET	/{content-type}/:id	Get a specific {content-type} entry
GET	/{content-type}/count	Count {content-type} entries
POST	/{content-type}	Create a {content-type} entry
DELETE	/{content-type}/:id	Delete a {content-type} entry
PUT	/{content-type}/:id	Update a {content-type} entry

Discovery Service

It allows automatic detection of network locations for service instances, which could have dynamically assigned addresses because of auto-scaling, failures, and upgrades

Gateway Service

Provide a proxy (routing) and client-side load balancing via ribbon You can deploy multiple services for the same service and gateway will load balancing between them (Simple scalability) Details: Spring Cloud Gateway

Online Payment

Support PayPal, Stripe integration

Search

Flexible domain search capabilities in Modern-Ecommerce are provided through integration with Elasticsearch

Product Service

Method	Path	Description	User Authenticated	Role
GET	/list?page={page}&pageSize={pageSize}	Get all products	x	Any
GET	/list/merchant?page={page}&pageSize={pageSize}	Get products created by merchant	x	Merchant, Admin
POST	/save	Create new product	x	Merchant
PUT	/save	Update existing product	x	Merchant
GET	/list/available?page={page}&pageSize={pageSize}	Get available products with inventory > 0	×	Any
GET	/list/not-available?page={page}&pageSize={pageSize}	Get available products with inventory = 0	×	Any

Order Service

Method	Path	Description	User Authenticated	Role
GET	/list/user?page={page}&pageSize={pageSize}	Get login user orders	x	Any
GET	/{orderId}	Get order by id	x	Any
POST	/save	Create new order	x	Client
PUT	/save	Update existing order	x	Client

Product Catalog Service

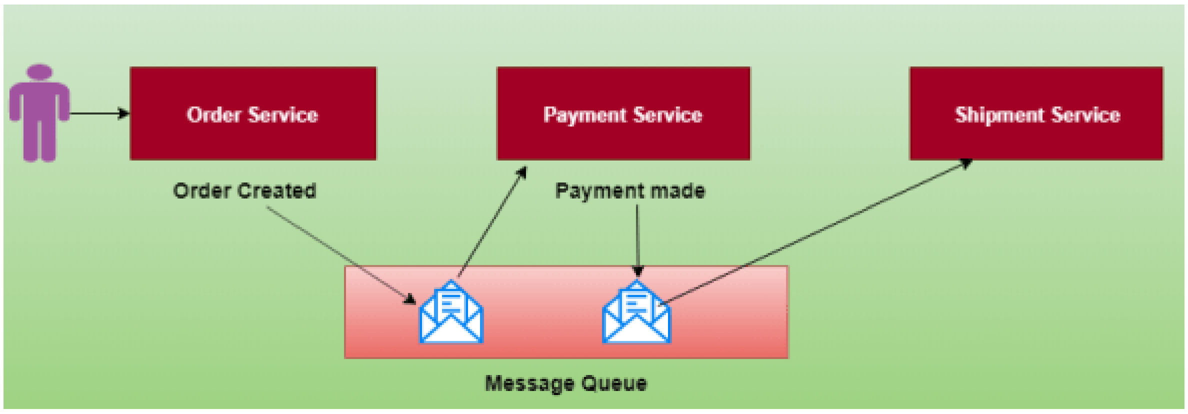
Method	URI	Description	Parameters	Request JSON	Response JSON
GET	*/products/recommendations*	List of recommended products	None	*[TODO]*	*[TODO]*
GET	*/products/{id}*	Fetch product information based on id	None	*[TODO]*	*[TODO]*
PUT	*/products*	Adds new product	*[TODO]*	*[TODO]*	*[TODO]*
POST	*/products/{id}*	Updates existing product	*[TODO]*	*[TODO]*	*[TODO]*

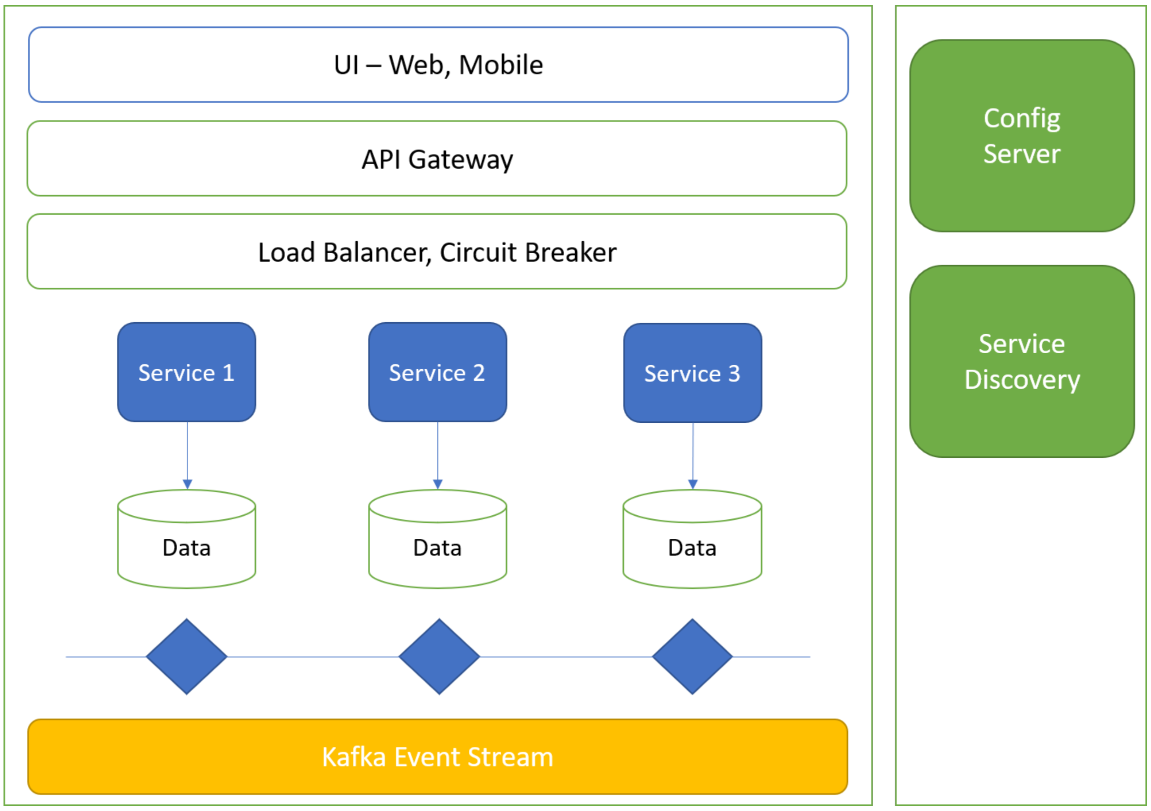
Cart Microservice

REST API

Cart REST API supports following operations,

Method	URI	Description	Parameters	Request JSON	Response JSON
GET	*/cart/{id}*	Fetches cart by id	None	*[TODO]*	
POST	*/cart/{id}*	Creates or updates cart	*[TODO]*	*[TODO]*	


Order Service
Payment Service
Shipment Service
Order Created
Payment made
Message Queue


UI – Web, Mobile
API Gateway
Load Balancer, Circuit Breaker
Service 1
Service 2
Service 3
Data
Data
Data
Kafka Event Stream
Config Server
Service Discovery