Solution

Gathering the requirements

Functional requirements

- Create User Profile
- List Products
- Search Products
- Add/Modify Products
- Purchase Products
- Order Tracking
- Order History
- Product Recommendations
- Product Reviews

Non-functional Requirements

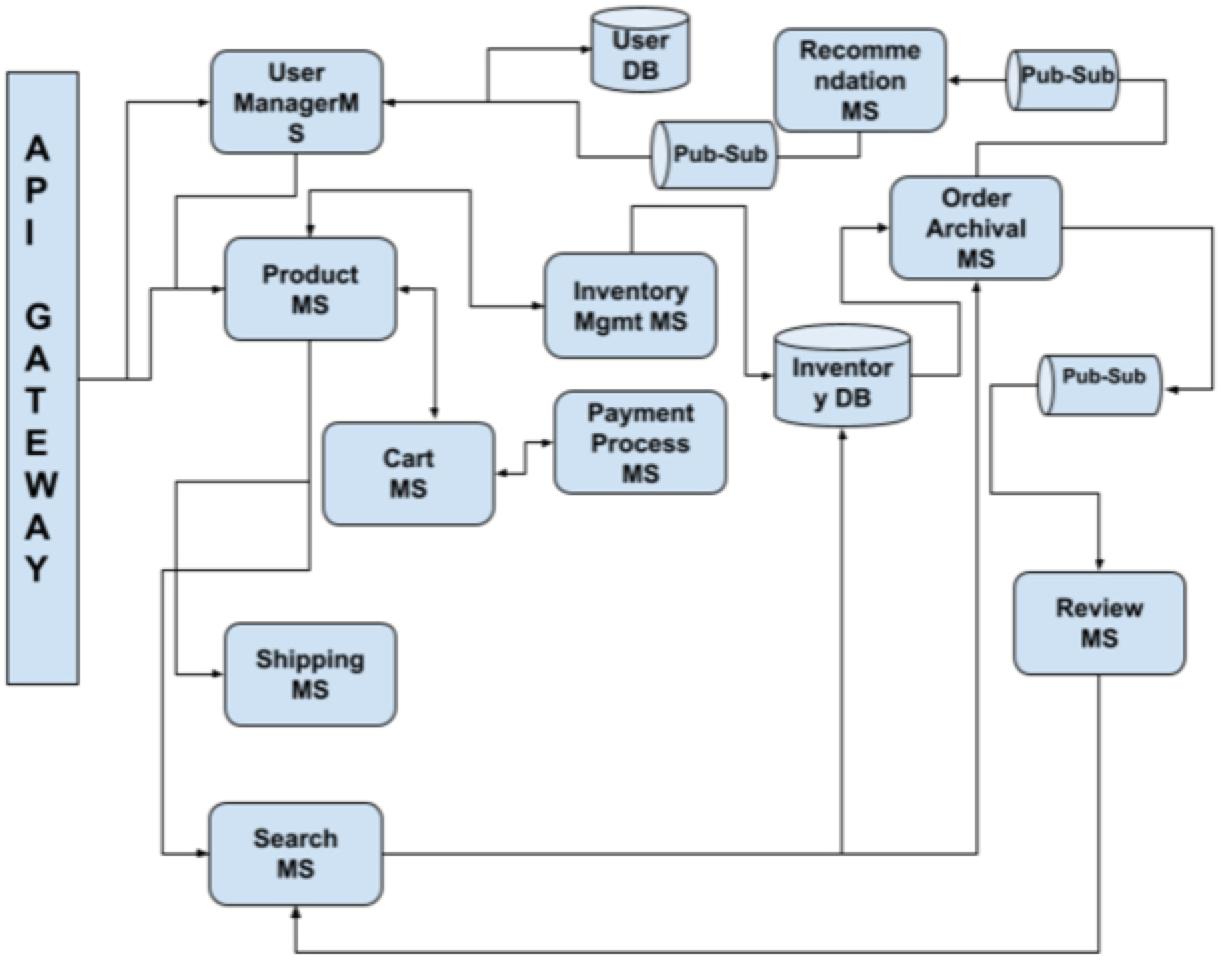
- Low latency
- High availability
- High consistency

Identifying microservices

Following microservices can be part of the eCommerce application based upon the functional requirements identified.

- User Manager Microservice
- Products Microservice
- Search Microservice
- Cart Microservice
- Order Archival Microservice
- Payment Process Microservice
- Recommendation Microservice
- Review Microservice
- Inventory Management Microservice
- Shipping Microservice

Logical architecture



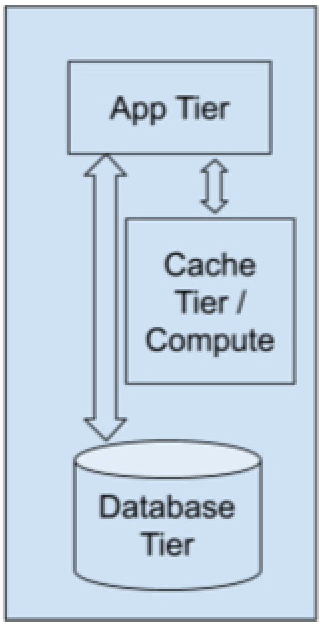
Detailed Design of Microservices

Microservice	Type of Technology
UserManager Microservice	K-V Workloads, Simple Data Storage on Disk and Hash Table K-V in memory
Products Microservice	K-V Workloads, Simple Data Storage and Access can be CRUD and In memory Cache
Search Microservice	Compute Intensive Workload, In memory Graph
Cart Microservice	K-V Workloads Simplate Data Storage and Access data

	can be CRUD
Shipping Microservice	K-V Workload,
OrderArchival Microservice	Compute Intensive Workload, Simple Data Storage on Disk
PaymentProcess Microservice	K-V Workload Compute Intensive Workload, In-memory Processing, with Simple Data Storage for payment details for the transaction
Recommendation Microservice	Offline Compute Intensive, Online: Streaming Analytics Workload
Review Microservice	Streaming Analytics workload with Simple Data Storage and Access
Inventory Management Microservice	Compute Intensive Workload, with Simple Data Storage on Disk

UserManager Microservice

This microservice is responsible for accepting requests from the User for account creation, account update / modification for password, address, phone number. Also this microservice is used by the user to login into the e-commerce site for placing orders as well as for querying the status of the orders.



Data Design

a. createUserAccount(userName, email, password, phoneNumber, address, country)

This method creates the user account for the first time. It stores the information in a simple K-V workload CRUD based database. On successful creation of the account, a unique userId is generated and stored against the created user account. The database could store the uniquely generated userId to index into the database for faster lookup and information retrieval.

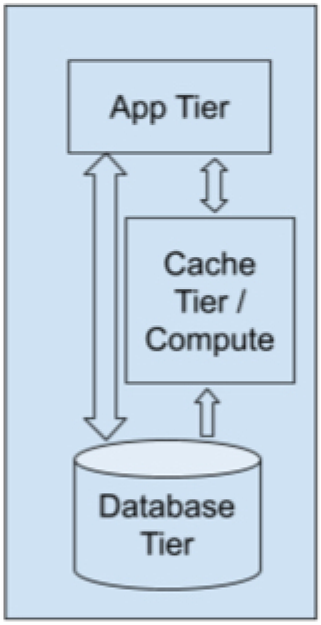
b. updateUserAccount(userName)

This method is used to modify any of the fields except the userName and email address combination to avoid adding duplicate records in the database.

UserName	Email	Password	PhoneNumber	Address	Country

Products Microservice

When the user accesses the website of the ecommerce application, a default home page is displayed. Here the user can either login with his account credentials or continue to browse the home page. The user could search the products that he/she is interested in purchasing from the site. As this could be read-heavy operation, the cache tier can maintain the products that have been accessed by this user. This microservice is a simple K-V workload with simple database access.



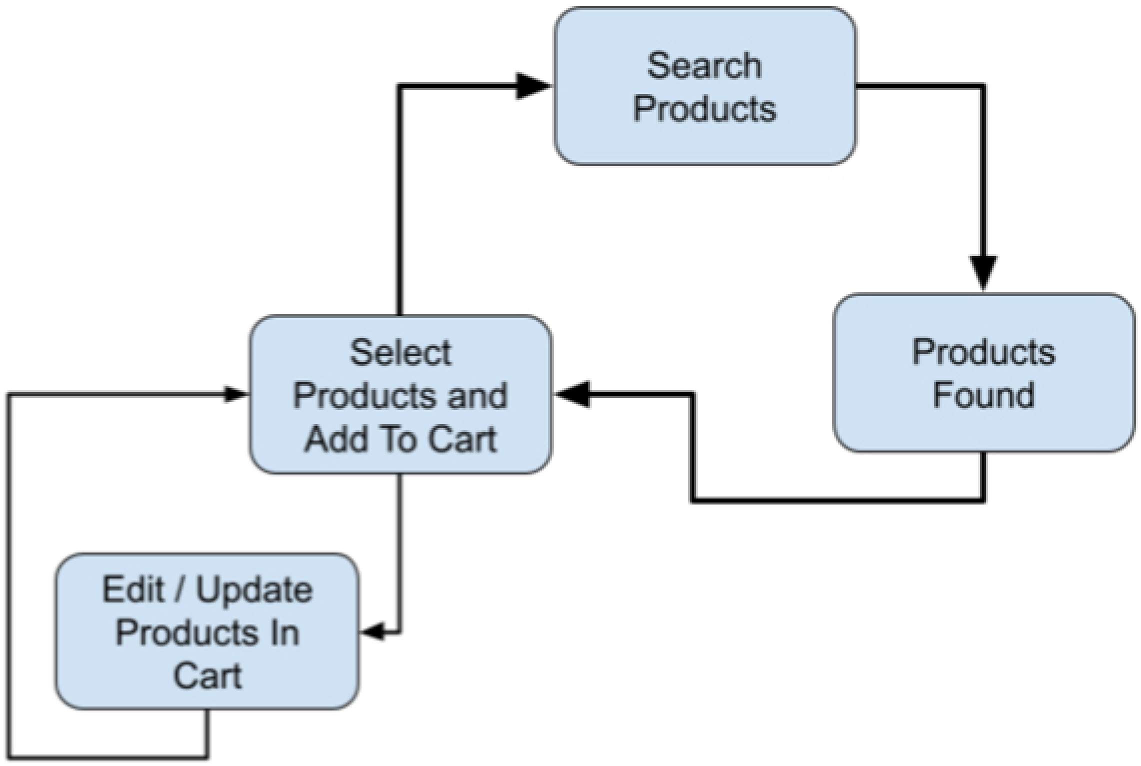
Data Design

a. List<Products>showProducts(searchString)

This API returns all the products based upon the search string that this user entered. The search results are obtained from the database. These results are cached in the cache tier also. So when the user searches the product again, the results are obtained from the cache instead of querying from the database.

b. List<Products>showProductsFromCache(searchString) from cache
This method is returned from the cache tier itself, if a matches the showProducts query criteria being matched. The cache tier has a Time-To-Live (TTL) field associated with each such record and it gets evicted on expiry of the TTL.

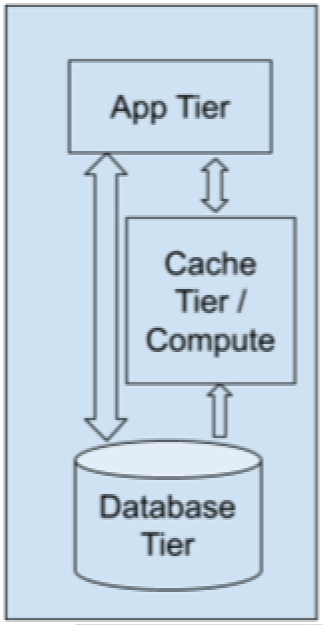
c. selectProducts(userId, List<Products> products)


Search Products
Products Found
Select Products and Add To Cart
Edit / Update Products In Cart

d. editProducts(userId, List<Products> products)

Search Microservice

This is one of the heavily used microservice on the e-commerce platform / application. The user can use this service without logging onto the platform or update / modify / edit the products that have been added to the cartMicroservice. This is a computer intensive workload.



Data Design

a. List<Products>(userId, searchString)

This API will be invoked by Product Microservice to search products for editing, updating the products added to the cart.
This API will also be invoked from the e-commerce application home page, when a user logs in for browsing the products available on the platform. These searches are stored in the database, and are available the next time when the user logs back in or returns to the home page of the e-commerce application.

UserId	Products Searched	Search Date

With the SearchDate being stored, it will act as TTL for the search listing. This information can be automatically purged from Cache Tier or Database Tier, based upon the user activity.

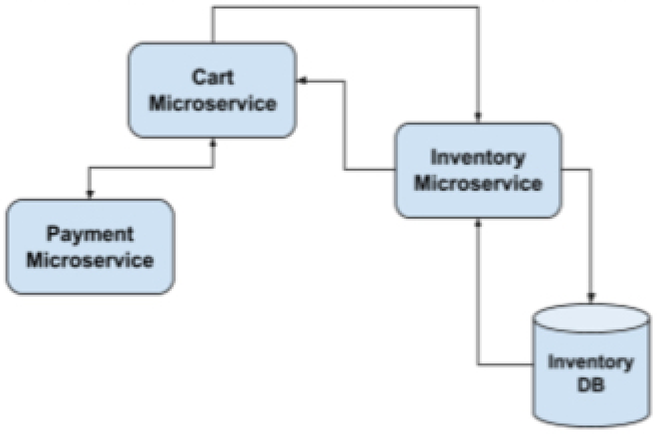
b. List<Products>(searchString)

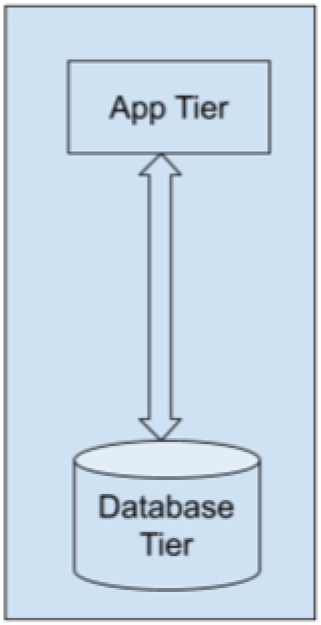
This API will be invoked when the user browses various products from the e-commerce application home page. The results of the search will be displayed as a search outcome. This search history will not be associated with the user and will not be stored either in Cache Tier or Database Tier.

Cart Microservice

When the products that were selected by the user get added to the cart. When the user initiates to purchase the products, a check is made through the inventory microservice if the products are available for the products added to the cart. Two forms of payment options are possible:

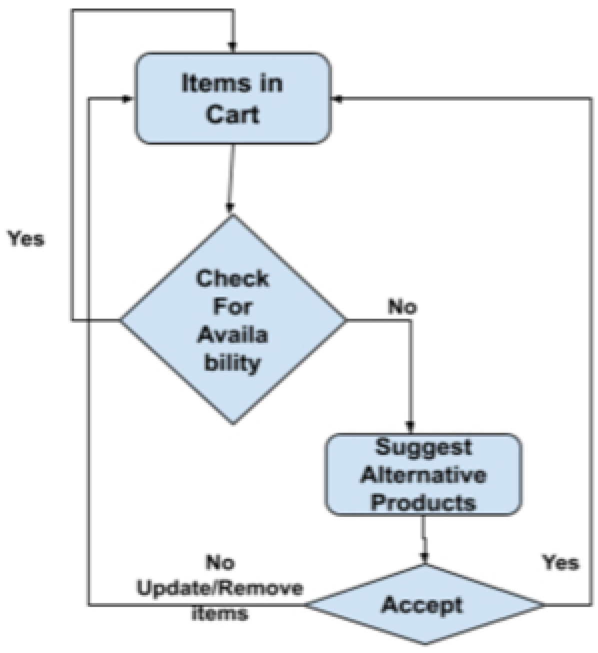
a. As part of UserManager Service, to use the payment details that are already added:
    i. Credit Card Payment
    ii. Debit Card Payment
    iii. Gift Card Payment
b. Or Add any of the above payment details for this particular transaction.
c. The Cart Microservice initiates a request to the Payment microservice, and either the transaction can be successful or failure is possible.



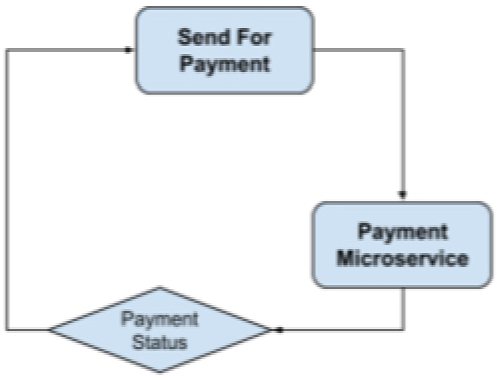


Data Design

a. getCartItemsAvailability(userId, <List> Products)
Before the items that are present currently in the cart, be purchased, we need to ensure that the inventory is available for these items. A request is sent to the Inventory Microservice to ensure that the items are indeed available for purchase. On successful verification the next phase of transaction is to send information to the Payment Microservice. If some are unavailable, then the user is notified that either these be removed or suggest alternate items that could be substituted.

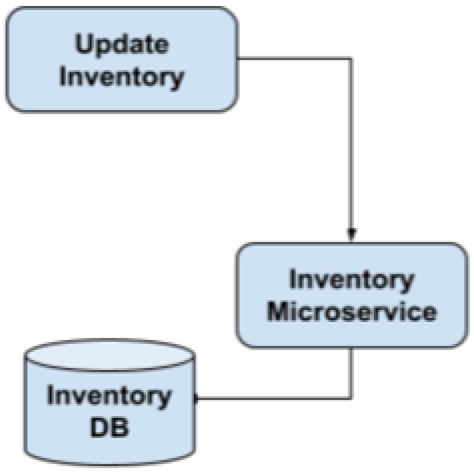


b. sendPaymentRequest(userId, amount)
   The CartMicroservice initiates an API call to the PaymentMicroservice for the amount that needs to be charged for the items that were requested.

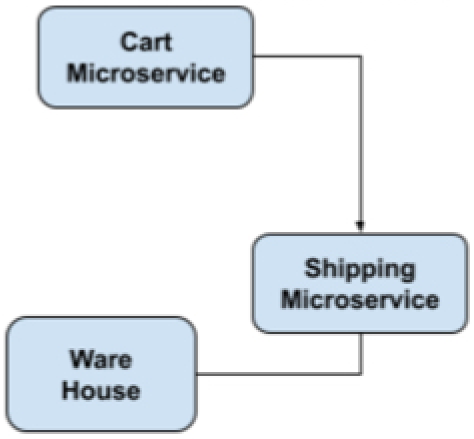


   The payment status can be either success or failure. There cannot be partial payments that can occur. The return value on success will be a transactionId. This value is stored in the OrderArchival database.

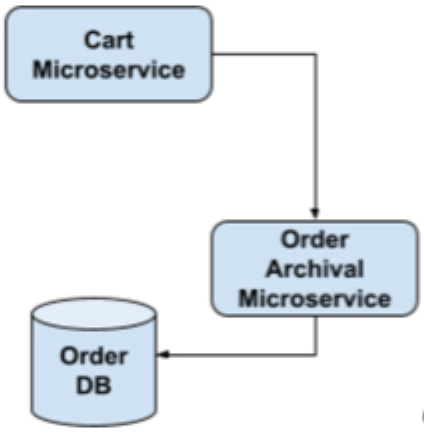
c. updateInventory(<List> Products)
   After the payment to the selected products is successful, the updateInventory method is invoked to notify the Inventory Microservice.



d. notifyShippingInfo(userId, address, shippingSpeed, <List> Products)
The Cart Microservice notifies the Shipping Microservice about the products to be shipped to the address that has been passed. Based upon the shipping speed requested, the shipping microservice fetches the items from the nearest warehouse and initiates shipping to the address provided.



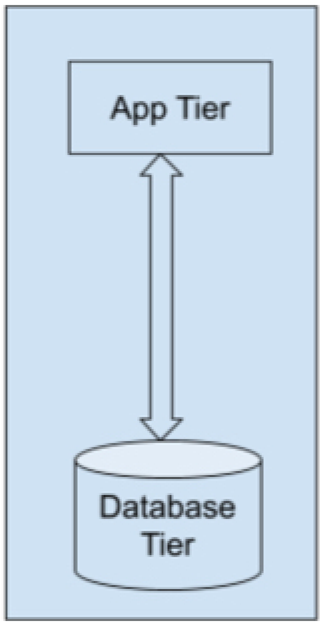
e. updateOrdersToArchival(userId, <List> Products, orderId, shippingId, paymentId,



OrderStatus, orderDate)

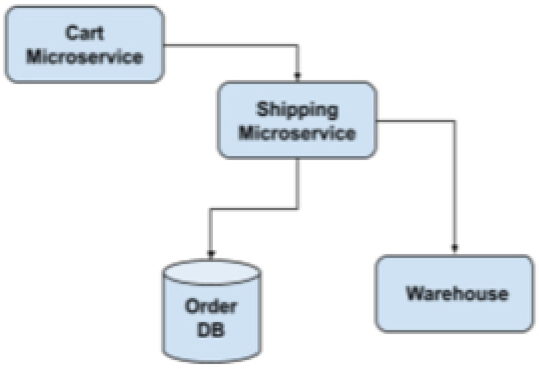
Shipping Microservice

When an order successfully goes through the Payment microservice, the Cart microservice invokes the notifyShippingInfo API towards the Shipping microservice. Based upon the type of shipping speed chosen at the time of the order and the destination address, the items are collected from the warehouse and dispatched to the customer.



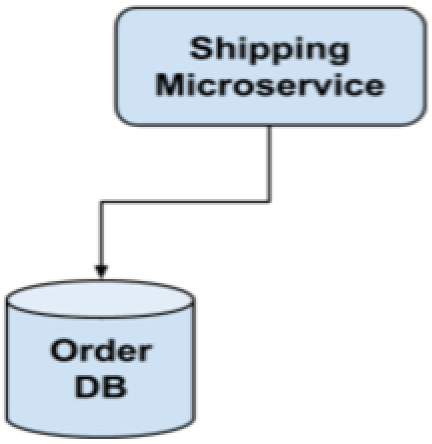
Data Design

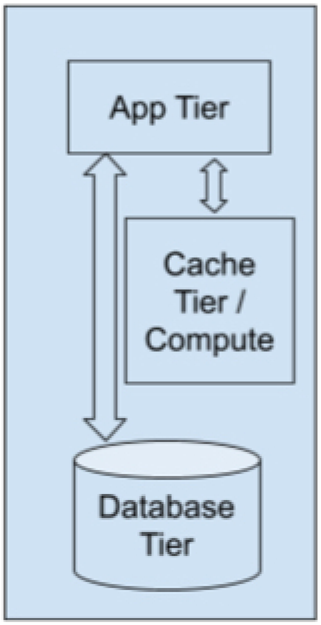
a. shipItemToCustomer(userId, shippingAddress, shippingSpeed, <List> Products)



b. updateOrderStatus(userId, OrderId, shippingStatus, itemReturned)

This API updates the order status in the OrderDB with the shipping status, when completed and delivered. This API will also be used when a customer initiates a return of a particular item part of the order.





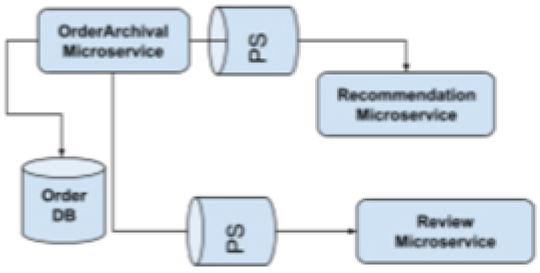
OrderArchival Microservice

The OrderArchival microservice will receive updates from Cart Microservice when a customer places an order and successfully payment is applied. The Shipping microservice will also update this microservice about the order shipment status. This is a Compute Intensive workload when publishing to the Recommendation microservice as well as to the Review microservice.

This microservice through pub-sub stream processing publishes to the Recommendation microservice, as well as Review microservice.

Data Design

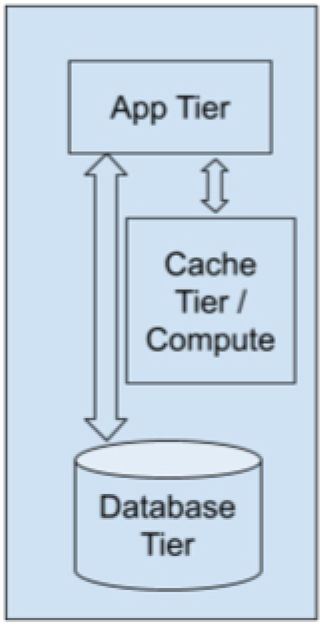
a. publishOrderInfoToRecommendationService()



b. updateReviewInfoToReviewService(orderId, <List> products, reviewString)

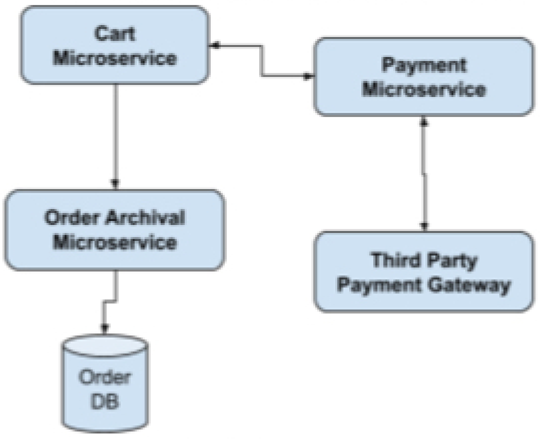
PaymentProcess Microservice

This microservice interfaces with external 3rd Party payment gateway for processing the payment info received through the Cart microservice.

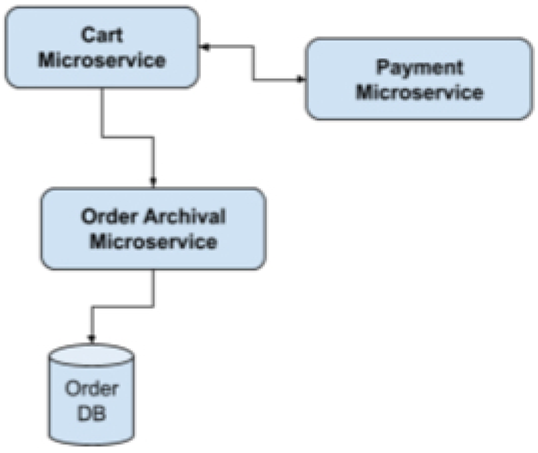


Data Design

a. getAuthorizationFromPaymentGateway(amount)



b. updateOrderWithTransaction(userId, orderId)

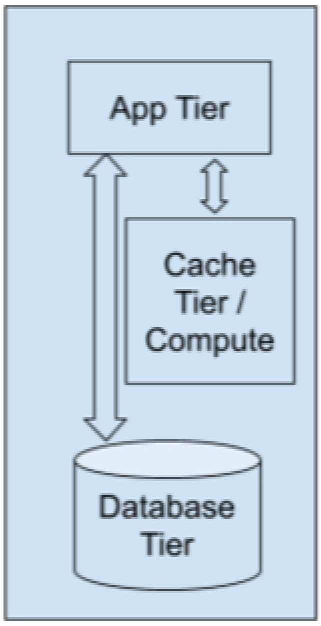


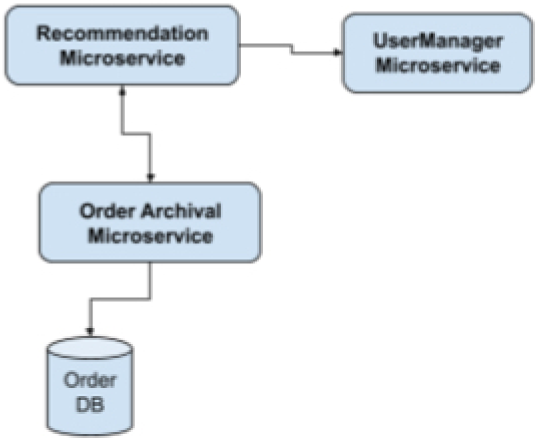
Recommendation Microservice

The recommendation microservice is one of the backend microservice that is a computer intensive workload. This retrieves the order information from the OrderDb and provides suggestions / recommendations to the user.

Data Design

a. createRecommendationFromOrderArchival(userId, orderId, <List> products, ttl)
   This method retrieves the orders that have been placed by the specific userId, and provides recommendations. The in-memory data would be stored in hashMap for constant lookup.



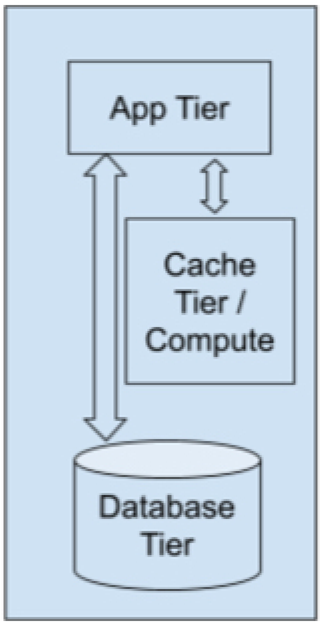


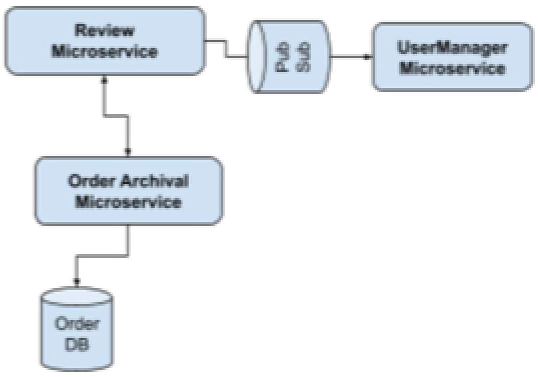
Review Microservice

The ReviewMicroservice is also one of the compute intensive workload services.

Data Design

a. String getReviewForProduct(<List>userNames, productName)



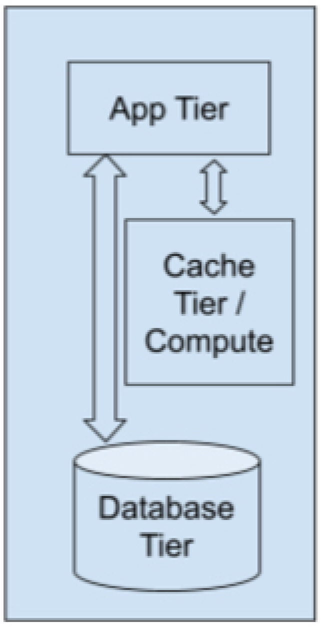


This method gets the review comment provided by users for specific products.

b. updateReviewForProduct(userId, productId, productName, reviewComments, reviewDate, reviewRating)

InventoryManagement Microservice

This is one of the K-V workload based services. This interfaces with third-party sellers on the e-commerce platform to update the inventory that is available.



Data Design

a. <List>Products getInventoryInfoByProductId(productId)
b. <List> Products getInventoryInfoByPopularity()

Need For Scale

For the microservices outlined above, the common bottlenecks that needs to be addressed include:

a. Storage
    i. User DB
    ii. Inventory DB
    iii. OrderArchival DB
b. Throughput
    i. Search Microservice

   ii. PaymentProcess Microservice
   iii. Cart Microservice
   iv. Shipping Microservice
c. Availability
   i. Products Microservice
   ii. Cart Microservice
   iii. PaymentProcess Microservice
   iv. Shipping Microservice
d. Hotspots Removal
   i. Inventory DB
   ii. OrderArchival DB
   iii. User DB

Proposed Distributed Architecture

As outlined in the 'Need For Scale' section, based upon whether it is K-V workloads or Compute Intensive workload, the distributed architecture needs to be identified.

Sharding

The User DB, Inventory DB and OrderArchival DB needs to be sharded horizontally with K-V pairs.
User-DB

User-ID (Key)	Name	Address	Email	Phone#

InventoryDB

ProductId(Key)	Product Name	Product Quantity	Minimum Quantity	Product Qty in Use

OrderArchival DB

UserId(Key)	ProductId (Key)	Product Name	Quantity Purchased	Date Of Purchase	Shipping Status

Replication

All the above DBs identified in the Sharding section, have to be replicated for both Availability as well as for throughput.

Applying CAP Theorem

Data Storage	CP or AP
User DB	CP
InventoryDB	AP for Querying CP for Order Fulfilment
OrderArchival DB	CP