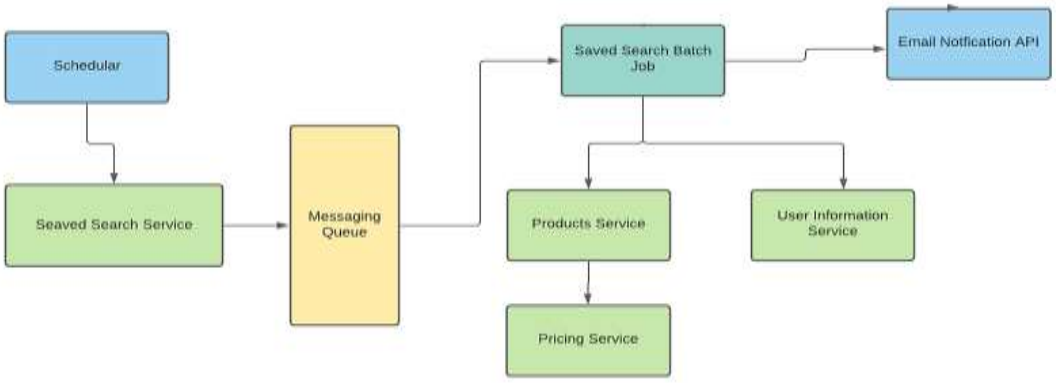
Saved Searches Services/APIs are commonly used these days in many applications. They are extremely beneficial to facilitate customers to receive notification of a specific product or the availability of a product based on search criteria that they have specified.

To design the solution of the given problem, there could be multiple ways, but we could implement hybrid approach, and implement the solution using batch job API along with micro-services based architecture.

The batch job would run periodically to check if notification for user's saved search is required to be sent. We can construct the saved search feature by implementing few micro-services which can work independently to provide the desired results.

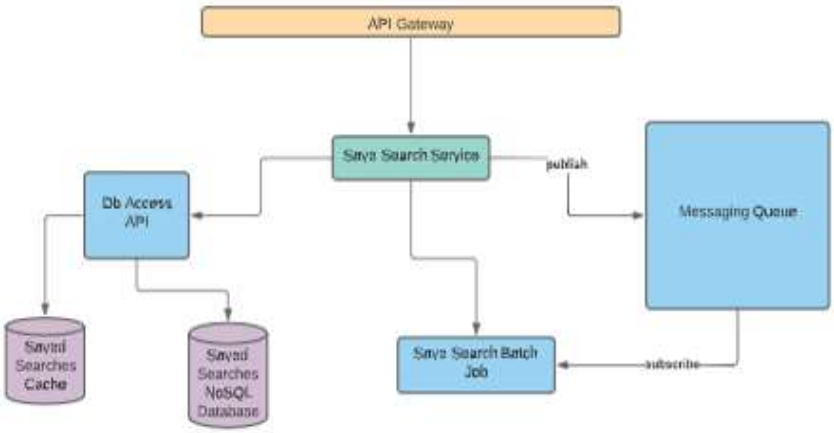
Following high level design depicts the interaction of different micro-services and downstream APIs to build this functionality



High level design (MSA)

Saved Search Service

Saved Search Service would allow the Web or Mobile Apps users to save their searches. In addition, we would implement internal API in this service which would be used by scheduler to fetch the all those saved search whose notifications are required to be sent. Saved Search would fetch the records from DB and trigger the email notification process with message queues.



Saved Search Service HLD

Following would be Rest API specifications for critical endpoints

API	SavedSearchesService
Base Path	*/userSearches*
Dependency	Batch Job API, Messaging Queue, and Emai Notification API
Functionality	*Save User search in the data store to trigger an alert notifications*
Verb	POST
Resource	*/save*

Save Search Rest API Specification

API	SavedSearchService
Base Path	*/userSearches*
Dependency	
Functionality	*Get all the Saved Searches bu the user*
Verb	GET
Resource	*/userId*

Get Saved Search Rest API Specification

- **Implementation**
  - This can be a RESTFul service with any supported language such as Java or C# with a NoSQL data-store.
  - It will provide endpoints to create, retrieve and delete for saved searches.
  - It will also provide endpoint for scheduler to invoke saved search notification to user.

  The endpoints API functions would look like these:

  - **saveSearchNotifition(searchCritera, userID)**
  - **deleteSearchNotification(APIKey, NotificationID)**
  - **retrieveSearchNotifications(userID)**
  - **sendNotifications(userID, productSearchCritiera)**

- **Database Design/Model**

  The database schema for user saved search would include the following significant fields
  ID – Auto generated ID (GUID)
  UserID – User for which saved search entry is persisted

Frequency – frequency of alerts such as weekly, monthly etc.

Following would be sample of complete data object for saved search

```
{
  "userSavedSearch": {
    "id": "userSavedSearch-01",
    "userId": "user-01",
    "frequency": "weekly",
    "searchKeywords":"BMX Z4",
    "category":"category-01",
    "subCategory":"subCateogory-01"
    "priceFrom":"2000",
    "priceTo":"4000",
    "searchExecutionDate":"19-12-2020"
    "lastAlertsNotificationDate":"19-12-2020",
    "nextAlertNotificationDate":"26-12-2020",
    "products-listing":10,
    "active": true,
    "emailNotificationsEnabled":true,
    "pushNotificationEnabled":true,
    "smsNotificationEnabled":true
}
```

User Saved Search NoSQL Document

Saved Search Batch Job

Saved Search Batch Job can be implemented in separate downstream client library which is invoked by Saved Search Service using messaging queue. The reason for using queue to implement control the number of requests which needs to be processed at the particular time period to ease the load on the infrastructure and increase the performance.

Once the request is reached to Batch Job it will invoke the Product Service to get the details about the product listing and also in parallel, call can be send to invoke User information Service to fetch the customer details.

After collecting all the data and transforming the request for email notifications, it would make asynchronous call to Email Notification API to process the batch of email notifications.

- **Implementation**

  - This can be implemented as downstream client library with any supported language such as Java.
  - This library will provide the public interface to send notifications in batch for multiple users

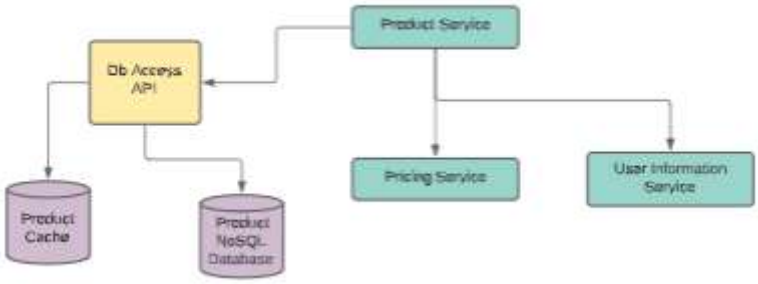
Product Service

Product Service can be build as independent micro-service, and can be useful for other features as well, the core functionality of this service in this feature to provide the product listing based on

saved search criteria, which can be product title, keywords, category, sub category or even the location.

In this feature, once Rest endpoint would be used to get the product listing based on search criteria, the response of this endpoint would be the list of products with their complete details along with pricing, location and seller information.

To fetch all this information, product service can communicate with pricing service to get the current and history of price updates for the specific product. In addition, it needs to interact with the user information service to retrieve the details about seller.



Product Service HLD

Following would be Rest API specifications

API	ProductService
Base Path	*/products*
Dependency	
Functionality	*Get the Product Details*
Verb	GET
Resource	*/productId*

Get Product Details Rest API Specification

- **Implementation**
  - This can be a RESTFul service with any supported language such as Java or C# with a NoSQL data-store.
  - This service will provide endpoints to create, delete, search product based on criteria.

  The endpoints API functions would look like these:
  - **getProducts(productSearchCriteria)**

- **Database Design/Model**

  The database schema for product service would include different documents for parent product and also for associated entities such as product category, location etc.

Importantly below model provide the inner object of “Product Attributes” which suffice our needs to store the different type of products having different parameters or attributes.

Product document would look something like this

```
{
  "product": {
    "id": "product-01",
    "title": "BMW",
    "description": "Itinerary",
    "category": "prodCategory-01",
    "subCategory": "subCategory-01",
    "modified": false,
    "status": "Online",
    "active": true,
    "sellStartDate": "18-01-2021",
    "sellEndDate": "20-01-2021",
    "modifiedDate": "18-01-2021",
    "size": "340kg",
    "weight": "12",
    "seller":"seller-101",
    "location": {
       "town":"Jumariah",
       "state":"Jumariah",
       "city":"Dubai",
       "Country":"UAE",
       "region","Asia"
    }
    "productAttributes": {
      "mileage": "7500KM",
      "color": "blue",
      "usage": "average",
      "class": "Z Series",
      "make": "BMW",
      "model": "BMW z4",
      "horsePower": "2.5"
    }
    ]
  }
}
```

Product NoSQL Document

Category and **subcategory** documents would look something like this

```
{
  "category": {
      "id":"category-01",
      "name": "Vehicle",
      "modifiedDate": "17-10-2020"
  }
}
```

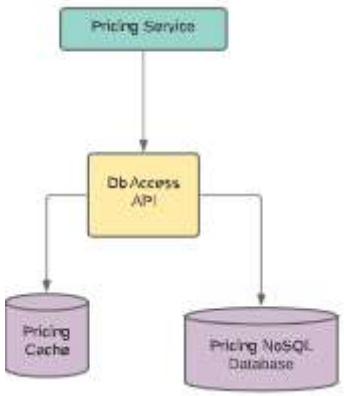
```
{
  "subpCategory": {
      "id":"subCategory-01",
      "name": "Sedan",
      "modifiedDate": "17-10-2020"
      "parent":":"prodCategory-01"
}
```

Location document/model can be constructed like following

```
{
"location": {
        "town":"Jumariah",
        "state":"Jumariah",
        "city":"Dubai",
        "Country":"UAE",
        "region","Asia",
        "postalCode":"0000",
        "pobox":"asd21"
    }
}
```

Pricing Service

Pricing Service would be another micro-service for this feature, which would be used to retrieve the pricing information including current price and history of price updates for the specific product. This service maintains its own data store documents for each product.



Pricing Service HLD

Following would be Rest API specifications

API	PricingService
Base Path	*/pricing*
Dependency	
Functionality	*Get the Pricing Details of the product or service*
Verb	GET
Resource	*/products/productId* */services/serviceId*

Get Pricing Details Rest API Specification

- **Implementation**
  - This can be a RESTFul service with any supported language such as Java or C# with a NoSQL data-store.
  - This service will provide endpoints to search pricing information of the product.

  The endpoints API functions would look like these:
  - **getPrcingDetails(productId)**

- **Database Design/Model**

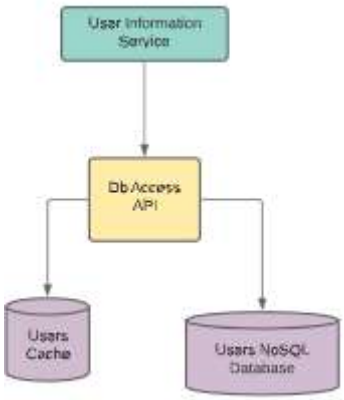
  The database schema for pricing service would include different pricing document for each product. Following NoSQL document sample also provides the details on current and also the history of all the modifications in price

```
"pricing": {
   productId: "product-01",


    current:{
    "amount": {
      "value": 1099,
      "quotedValue": "1099.00"
      "startDate":"21-01-2021",
      "modifiedDate":"21-01-2021",
      "currency":"AED"
      }
    },

    history:[
    "amount": {
      "value": 1120,
      "quotedValue": "1120.00",
      "startDate":"17-01-2021",
      "endDate":"18-01-2021",
      "currency":"AED"
      },

      "amount": {
      "value": 1135,
      "quotedValue": "1150.00",
      "startDate":"18-01-2021",
      "endDate":"18-01-2021",
      "currency":"AED"
      }
    ]

}
```

Pricing NoSQL Document

User Information Service

User information service is the micro-service which would be used to store and fetch the information about the sellers and customers. As this service would have personal information, so it needs to secured service and would require token based authorization to access it.



User Information Service HLD

- **Implementation**
  - This can be a RESTFul service with any supported language such as Java or C# with a NoSQL data-store.
  - This service will provide endpoints to create, update, search register customers and seller in the system

  The endpoints API functions would look like these:

  - **getUserDetails(userId)**

- **Database Design/Model**
  The database schema for user information service would include different user document for all register customers and sellers.

```
{
  "User":{
    "id":"user001",
    "userId": "user001",
    "active" true,
    "location":"local-01",
    "isSeller":true
}
```

Following would be Rest API specifications

API	UserInformationService
Base Path	*/users*
Dependency	
Functionality	*Get the User Details*
Verb	POST
Resource	*userId*

Retrieve User Information Rest API Specification

Email Notification API

The Email notification API can be built using JavaMail API with Java or mailing API with C#. This API can be added to any calling service as dependency in the service.

Technical Suggestions

- Service can be deployed on any Cloud Environment such as AWS, and different nodes can be deployed in AWS regions
- Services can communicate with each other via internal network routes such AWS DirectConnect or via Event streams
- API Gateway is required to manage the timeout and provide authorization for external users.