**TSD - CMS Microservice**

## **Overview:**

This is TSD of a campaign management system, Includes CRUD for setting up campaign, campaign Rules, Target groups linked to campaigns.

Customers to be added to target groups via admin panel, and groups can be added too.

Separate 2 endpoints for-

* Requesting eligible campaigns via customer & cart params
* Applying discount out of the requested campaigns as above. On applying we keep track of analytics as required.

**Tech:**

Django Rest - api

Redis - Cache, Broker for Celery for async processing

Postgres

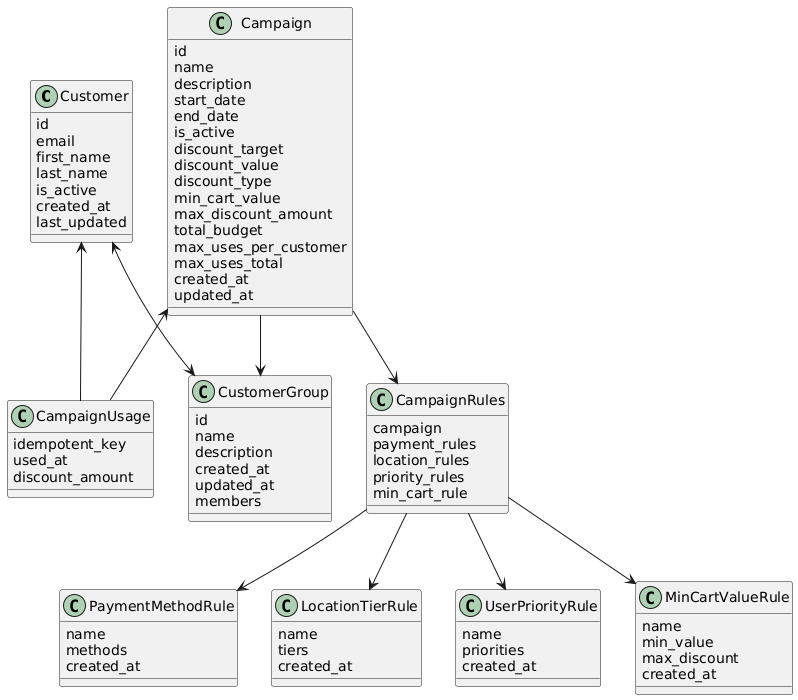
Celery - scheduled tasks, async tasks

Swagger - api docs

## **Data Model:**

We have Entities as declared in below UML diagram.

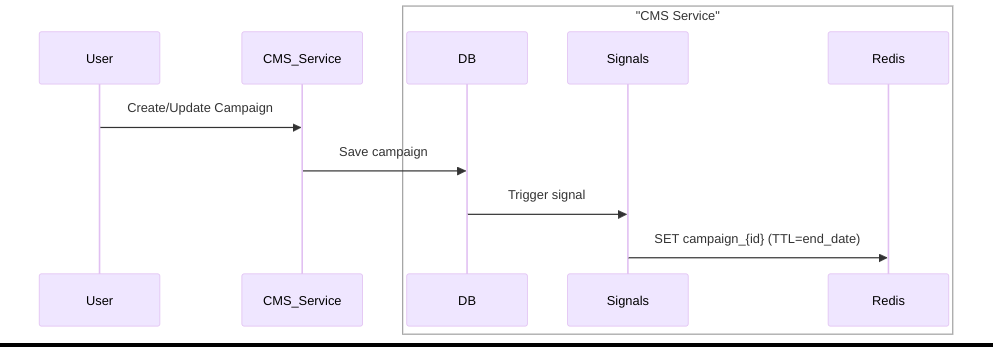
* Customer - have customer id as uuid, these are the customers which will be querying our service.
* Customer Group - Groups have set of customers, groups have mapping to campaigns. So we can target certain group of customers for a campaign. 1 campaign can be associated to many groups.
* Campaign - Campaign Entity contains mostly static data, As this will server as source of truth for loading cache. Rules for checking the eligibility are linked directly to Campaign.
* Campaign Rules- this table has 1 on 1 linkage to campaign and many number of rules to be validated for eligibility of campaign/discount. Liked to as below set of rules
  + Priority
  + Location
  + Cart value Related
  + Payment Method
  + Can be extended with other rules as and when required
* Campaign Usage - This stores the data for actual utilization requested by any user for any campaign and request/discount metadata.



## **Architecture & Flow Diagrams:**

* **Creating Campaigns:** Campaign creation requires creation individual rules. These rules are linked to Campaign Rules, and campaign rule is linked to campaign.

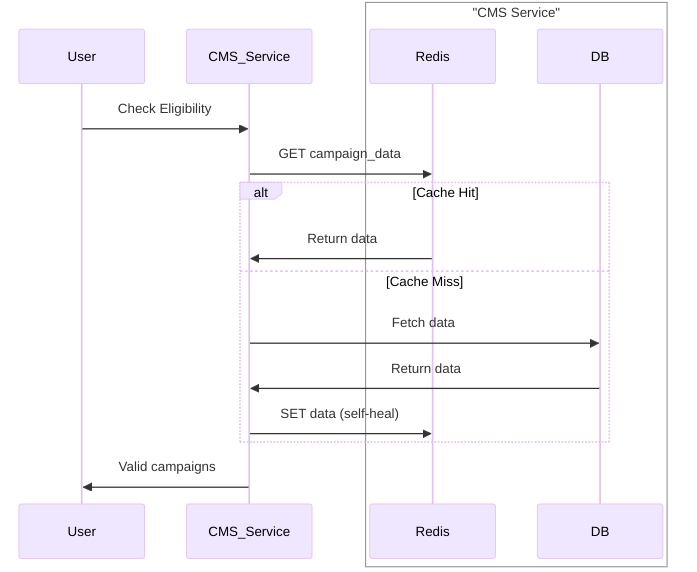
Create Campaign → Create/Reuse Rules (Location, Priority, Payment) → Link Rules (CampaignRules Table) → Activate

* **Adding Users/Groups:** Users can be added in bulk. (Functionality not added currently, future enhancements). Each group can have multiple groups.  
  Both Groups and users currently can be added via admin panel.  
  Groups are created as they are linked to campaigns for target certain customers within groups.
* CRUD Flow Diag - 
* Maintaining Campaign data in Redis Cache as in above diagram:

We have campaign data in our DB, keeping in mind the scale of our system, if for every request to check eligibility we query the database it would increase the load on our system. As data is mostly static once a campaign is created we maintain cache for each campaigns and list of active campaigns in cache.

* We have created signals for campaign table, as campaign is created/edited/deleted. This signal refreshes the cache for edited campaign. Hence making sure we have latest campaign data in cache. Campaign cache has TTL of its end date.
* We have a scheduled celery task to refresh cache every midnight. This makes sure we load any campaign created with future date into cache at midnight.
* On every service restart too we reload the cache.
* Flow Diagram for Requesting Eligible Campaigns: For requesting eligible campaigns, we hit cache to fetch all active campaigns from cache, if cache miss we query DB and reload cache. Using campaign data from cache we run rules check on provided customer and cart params in the request. Eligible campaigns are sent in response.

Below shows the request flow for same.



* Flow for apply discount - User gets all the data for eligible campaigns from above response. On apply discount call user sends in single request with campaign id and discount value & idempotent key to be utilised.

Idempotent key is used to track duplicate requests via cache. This key is a unique key per transaction generated on the client. Client is expected to send the same key for the same request.

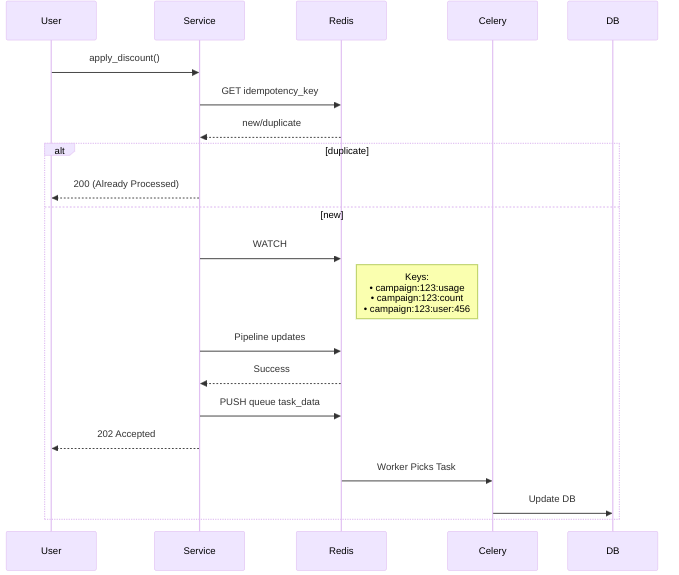
We maintain keys in 3 cache keys for campaign consumption tracking as described below-

* + Customer per day count key - This stores customer usage of a campaign for single day. This is used to keep track of single day usage per user per campaign & check that we have not surpassed daily per user limit.
  + Total count per campaign - used to keep count of total users who availed that campaign discount & to check we are not surpassing the total count limit.
  + Current budget usage - to check and maintain the budget consumed out of available budget per campaign.

These keys are storeed in cache, Currently fallback is not implemented but these can be populated from Campaign Usage Table in case Failure or data loss in cache. (This is not currently implemented, Future Enhancements)

On receiving the request we update these 3 keys in redis using pipeline to make sure all 3 are updated simultaneously and we do not face race condition/ contention when we are close to limit.

On successful Update an async task is called to update CampaignUsage Table in DB.



Note - Currently we are using Redis and celery async task which processes single request to make entry in DB. On future enhancements we will be moving to kafka and pushing the messages to kafka, as kafka works on log based system and is more reliable. We can also process in batches on Future enhancements instead of processing for every request. We can have a batch processing on kafka which will process from last processed data.

## **API Endpoints:**

For complete api endpoints and request response structure refer to swagger.json in repo (/cms/docs/swagger-api-docs.json) or swagger url on server (<http://127.0.0.1:8000/swagger/>).

We have CRUD for -

* **campaign**
* **campaign-rules**
* Rules [**Location**, **Priority**, **Cart Value**, **Payment Method**]

**Eligibility API** - Get - eligible-campaigns

To fetch all eligible campaigns based on cart and customer param rules check.

**Discount API** - Post - apply-discount

Applies the discount and saves data in db for transaction. Also keeps track of/ update the consumption stats per user, per campaign for count and budget consumption.