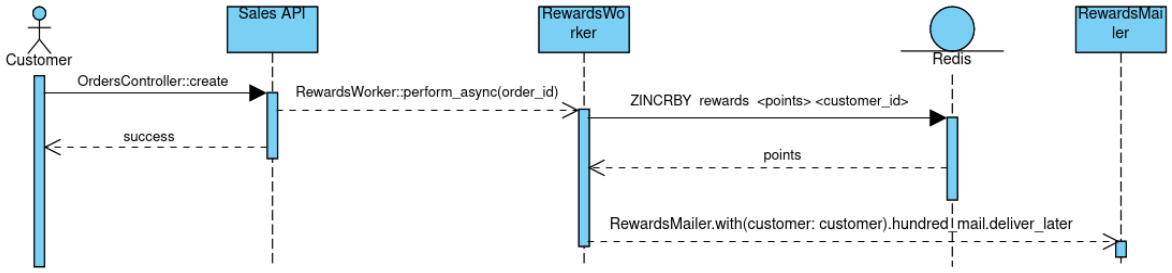
The Sequence diagram



Starting the process

Considering a Ruby on Rails application, with a MVC structure and also an API to handle mobile applications:

The main idea is to use a callback on the model responsible for receiving and processing the order. This way we can handle requests that came from the Web and also API requests. The callback would be invoked just after creating the order. Considering an Order model, we could think in a code like this:

```ruby
after_create :reward_order

def reward_order
  Reward::IncrementReward.perform_async(id)
end
```

The main goal is to process this in the "background" and respond quickly to the customer about the order result. The callback is invoking a Sidekiq worker, but we could consider any other method to handle pub/sub, such as RabbitMQ, SQS, etc.

And if the order creation process is handled by another service, such a service to handle payment gateway, this callback could be moved to an additional step into this other service.

Working in the background

Once the worker is invoked, there are some steps to follow here:

1. Get the instance of order using on informed id
2. Check the right number of points, based on the rules before defined
3. Increment the key of customers on Redis. This method returns the current score for the increment key, in this case, the customer_id. So, now, we have the current number of points for the customer.
4. Check if the number of total points is greater than 100. If it is, deliver an email to the customer.

Why Redis?

Redis is able to run in memory but also can be persisted. Other than that, Redis has the Sorted sets that make it possible for each element to be associated with a score and this score can be incremented by the number that we wish.

We could use any other storage structure, such as MongoDB or even PostgreSQL with a jsonb column, I just chose Redis because of the built-in sorted lists.