Hi Thiago!

Welcome to the 2017 Kisi Backend Coding Challenge. The coding challenge(s) below will be used to assess your familiarity with the Ruby programming language, the Rails framework and general coding best practice. There are no hard time limits, but we're trying to keep the scope limited so we don't occupy too much of your time. We will not use any of the results for our production systems. Our aim is to have an efficient and fair assessment: you spend time solving the challenge, and we spend time giving you detailed feedback.

Please choose **one** of the following challenges on the pages below and submit your results to a public Github repository, including a README.md with instructions on how to execute the code.

Challenge 1: Build a Ruby microservice that receives Event Webhooks from the Kisi API and notifies subscribed users via email based on rules

The Kisi API features an Event Webhook integration that calls 3rd party services to deliver events, e.g. based on user unlocks.

- ❏ Create a headless Ruby microservice (use a framework of your choice) that receives these webhooks
- ❏ The webhooks will go through a list of rules
- ❏ Each rule has a list of subscribers (array of email addresses is good enough)
- ❏ Each rule has a few fields that determine if the rule matches
    - ❏ Time: include time ranges during which the rule should be matched
    - ❏ User: include emails that should match (the event has an actor email field)
    - ❏ Action: which action should match, e.g. unlock
    - ❏ Object: which object type should match, e.g. Lock
    - ❏ Success: filter only successful or unsuccessful events (or both)

Demonstrate your microservice by filtering out unsuccessful unlock attempts and send an email to the subscribed users informing them that an unlock failed

Challenge 2: Build a background job system to use with Rails based on Google Pub Sub

The idea is to build a background job system that is compatible with the ActiveJob interface of Rails and allows Rails developers to easily enqueue jobs to a Google Pub Sub backend. The details:

- ❏ Transparent enqueueing with ActiveJob to Google Pub Sub
- ❏ Background workers dequeue job params and execute the corresponding jobs
- ❏ If a job fails, it should be retried at most two times five minutes apart
  - ❏ Three tries in total
  - ❏ Aggregate time between tries ten minutes
- ❏ Optional: Use ActiveSupport::Instrumentation to collect metrics
- ❏ Deliverable: Github repository with Rails project including CLI command to start the background job queue

Inspiration: https://cloud.google.com/ruby/getting-started/using-pub-sub (note: solution is lacking a mature OO approach and keeps the worker inside the adapter)

Challenge 3: Build a Devise strategy that authenticates Kisi links

Kisi links are optionally sent to users via email as part of access grants for doors. To use such links, the web app sends a custom Authorization header to the Kisi API.

- ❏ Create a Kisi account, place and door on https://my.getkisi.com
- ❏ Share a link access door with yourself
- ❏ Follow the link and intercept the calls from the web app to the API
- ❏ Build a Devise strategy that authenticates the user based on this header
- ❏ Pay attention to details: timing attacks, confirmation state of user and robust parsing of the Authorization header
- ❏ Include an exponential back-off strategy to prevent brute force attacks
    - ❏ What should determine when to back off?
    - ❏ Explain the pros and cons