Regular Ruby recruitment task

Background

We want to develop a ticket-selling platform. We are hoping to get popular pretty quickly, so prepare for *high traffic*!
While completing the test task please try to put yourself in the mind of the user as much as possible. It's very important to think of all the possible edge cases you can. The front-end part of the application is not yet ready, so feel free to design the API however you want.

High-level features

1. Get info about an event
2. Get info about about available tickets
3. Reserve ticket
4. Pay for ticket
5. Get info about reservation

Feature requirements

Get info about an event

1. Event has a name
2. Event has a date and a time
3. Event can have multiple type of tickets

Get info about available tickets

1. We should be able toreceive information about which tickets
2. are still available for sale and in which quantity.

Reserve ticket

1. Each ticket has a selling option defined:
2. `even` - we can only buy tickets in quantity that is even
3. `all together` - we can only buy all the tickets at once
4. `avoid one` - we can only buy tickets in a quantity that will not leave only 1 ticket avaiable
5. Reservation is valid for 15 minutes, after that it is realesed.

Pay for ticket

1. For the sake of simplicity we operate only on the `EUR` currency. Fee free to use
2. the provided adapter.

Get info about reservation

1. Return information about the state of the reservation and its data.

Additional info

1. We really don't want to push you in any certain direction, but if you've
2. made certain assumptions how it should work with FE then please let us know.
3. TL;DR - you can briefly describe how the whole app would interact.
4. Please use a relational (SQL) database such as Postgres, MySQL or SQLite.
5. Tests should cover happy path and edge cases you thought of.

```
# PAYMENT ADAPTER

# frozen_string_literal: true

module Adapters
  module Payment
    class Gateway
      CardError    = Class.new(StandardError)
      PaymentError = Class.new(StandardError)

      Result = Struct.new(:amount, :currency)

      class << self
        def charge(amount:, token:, currency: "EUR")
          case token.to_sym
          when :card_error
            raise CardError, "Your card has been declined."
          when :payment_error
            raise PaymentError, "Something went wrong with you
r transaction."
          else
            Result.new(amount, currency)
          end
        end
      end
    end
  end
end
```