Detallaremos a continuación los cambios que se realizaron

En el archivo REQ3:

- Renombramiento de la vista REQ3 por number_of_surveys_between_two_dates
- Renombramiento a variables firstdate y lastdate por initialDate y finalDate respectivamente.

En el archivo app:

- En el endpoint POST de responses se reemplazó el ciclo refactorizando con método de cierre de colección.

Código a refactorizar:

```
params[:question_id].each do |q_id|
    response = Response.new(choice_id: params[q_id], survey_id: survey.id, question_id: q_id)
    response.save
  end
```

Código refactorizado:

```
params[:question_id].map { |question_id| Response.create(choice_id: params[question_id], survey_id: survey.id, question_id: question_id)
```

- Cambio de nombres en el método get de surveys firstdate, lastdate por initial_date y final_date respectivamente.

- En el modelo survey:

Método **result** que pondera las carreras dada las respuestas de un survey y retorna una colección de estas ordenadas en orden descendente. El nombre será cambiado por **survey_result**.

También se refactoriza el siguiente **ciclo**:

```
career_weights = {}
careers.each do |career|
   career_weights[career] = 0
end
```

Refactorización:

```
careers.map { |career| career_weights[career] = 0 }
```

El siguiente ciclo anidado también será refactorizado:

```
responses.each do |response|                                    #
        response.choice.outcomes.each do |outcome|  #           # ciclo1
            career_weights[outcome.career] += 1     #ciclo 2    #
        end                                         #           #
end                                                             #
```

Ciclo 2, refactorización:

```
response.choice.outcomes.map { |outcome| career_weights[outcome.career] += 1 }
```

Ciclo 1, refactorización:

```
responses.each do |response|
            response.choice.outcomes.map { |outcome| career_weights[outcome.career] += 1 }
        end
```

A su vez ambos ciclos pueden ser factorizados por un método externo código a refactorizar:

```
responses.each do |response|
        response.choice.outcomes.map { |outcome| career_weights[outcome.career] += 1 }
end
```

Código refactorizado:

```
def evaluate_responses(careers)
  career_weights = {}
  careers.map { |career| career_weights[career] = 0 }
  responses.map { |response| response.choice.outcomes.map { |outcome| career_weights[outcome.career] += 1 } }
  career_weights
end
```

- En modelo careers:

Se cambia el nombre del método s_for_dates por number_of_surveys_between_two_dates.

- Luego el nombre de los parametros firstDate y lastDate serán reemplazados por initial_date y last_date.

- Luego las líneas en el método number_of_surveys_between_two_dates:

```
result = {}
result["name"]=name
result["number"]=0
```

serán reemplazadas por :

```
result = {}
```

- Se aplicó extract method en la siguiente condición:

```
Time.parse(initial_date.to_s) <= survey.created_at && survey.created_at <= Time.parse(final_date.to_s)
```

- is_created_at_between realizará la funcioń de comparar las fechas del survey asociado a la carrera con initial_date y final_date.

```
def is_created_at_between(initial_date, final_date)
  Time.parse(initial_date.to_s) <= created_at && created_at <= Time.parse(final_date.to_s)
end
```

- En el siguiente código, se aplicará método de cierre de colección:

```
surveys.each do |survey|
        result["number"] += 1 if survey.is_created_at_between(initial_date, final_date)
    end
```

Código refactorizado:

```
  result = { 'name' => name, 'number' => 0 }
  surveys.map { |survey| survey.is_created_at_between(initial_date, final_date) && result['number'] += 1 }
end
```

También se aplicará extract method en:

```
Time.parse(lastDate) <Time.parse(firstDate)
```

Por un nuevo método llamado **the_dates_are_valid?** retorna true si la fecha inicial si la primera fecha(initial_date) es menor o igual que final_date.

Código refactorizado:

```
  def the_dates_are_valid?(initial_date, final_date)
    Time.parse(initial_date.to_s) <= Time.parse(final_date.to_s)
  end
end
```