O objetivo desta etapa não é somente resolver o problema proposto. Mesmo que este problema não sejam extenso, nós esperamos que você encaminhe um código que acredite ser de qualidade, possível de ser executado e evoluído.

Use esse problema para nos mostrar as práticas que você utiliza no dia-a-dia, ou desejaria utilizar, para escrever um bom código. Ao mesmo tempo, cuidado para não cair em "*over-engineering"*.

Para a sua solução você poderá usar: Ruby, Elixir ou NodeJS.

Alguns pontos que avaliamos:

- design da solução

- legibilidade

- facilidade de evolução e manutenção da aplicação

- testes automatizados / uso de boas práticas de agilidade

- operacionalidade

- arquitetura da aplicação

- documentação

Você poderá usar bibliotecas de testes unitários ou ferramentas de build disponíveis para a linguagem que você escolher mas você não poderá utilizar outras bibliotecas com o propósito de resolver o problema.

Encaminhe uma breve descrição do design e suposições junto com o seu código, detalhes de implementação e arquitetura, bem como as instruções detalhadas de como rodar sua aplicação.

Queremos que nosso processo de seleção seja justo e as pessoas tenham as mesmas chances. Para isto, solicitamos que você não compartilhe ou publique nenhum destes problemas ou a sua solução.

O código-fonte da aplicação deve ser disponibilizado no github ou gitlab.

O candidato deve disponibilizar um link para sua aplicação publicada na internet. (Há diversos serviços grátis para isso: Heroku, Gigalixir e etc).

Como regra geral, lhe concedemos cinco (05) dias, partindo do dia em que você recebe o exercício, para criar esta solução. Caso você necessite de mais tempo, não hesite em nos solicitar. Se você tiver qualquer dúvida sobre o seu processo de seleção, por favor, entre em contato comigo.

Introdução ao problema:

A aplicação deve expor uma API, que irá receber o payload descrevendo a entrada e retornar a saída correspondente, conforme o exemplo dado. De acordo com o algoritmo e critérios de ordenação utilizados, os valores na saída podem ser diferentes do exemplo, não tem problema. Mas fique atento que a assinatura (a estrutura da resposta) deve ser mantida.

O código deve executar por si só.

Você deve prover evidências suficientes de que sua solução está completa indicando, no mínimo, que ela funciona utilizando os dados de teste que são fornecidos.

Gerenciamento de Palestras

Sua empresa está organizando um grande evento de programação, e recebeu muitas propostas de palestras para serem apresentadas. O problema é fazer todas elas encaixarem no tempo -- tem muitas possibilidades! Então você decidiu escrever um programa pra fazer isso pra você.

Será um único dia de conferência, mas ocorrerão muitas trilhas simultaneamente. Cada trilha tem uma sessão de manhã e outra de tarde. Cada sessão pode conter muitas palestras.

As sessões da manhã devem começar às 9 horas da manhã e terminar a tempo do almoço, que será servido às 12 (meio dia).

As sessões da tarde devem começar à 1 da tarde e terminar a tempo do happy hour.

O happy hour não pode começar antes das 4 da tarde, nem depois das 5 da tarde. O horário do happy hour deve ser o mesmo para todas as trilhas.

Seu programa pode considerar que não haverá uma palestra com números no nome.

A duração das palestras será dado em minutos ou com a string "lightning" indicando que ela durará 5 minutos.

Os palestrantes serão muito pontuais, você não precisa colocar um intervalo entre uma palestra e outra.

Entrada:

```
{
   "data":[
        "Writing Fast Tests Against Enterprise Rails 60min",
        "Overdoing it in Python 45min",
        "Lua for the Masses 30min",
        "Ruby Errors from Mismatched Gem Versions 45min",
        "Common Ruby Errors 45min",
        "Rails for Python Developers lightning",
        "Communicating Over Distance 60min",
        "Accounting-Driven Development 45min",
        "Woah 30min",
        "Sit Down and Write 30min",
        "Pair Programming vs Noise 45min",
        "Rails Magic 60min",
        "Ruby on Rails: Why We Should Move On 60min",
        "Clojure Ate Scala (on my project) 45min",
        "Programming in the Boondocks of Seattle 30min",
        "Ruby vs. Clojure for Back-End Development 30min",
        "Ruby on Rails Legacy App Maintenance 60min",
        "A World Without HackerNews 30min",
        "User Interface CSS in Rails Apps 30min"
   ]
}
```

(Possível) Saída:


```
{
  "data": [
    {
      "title": "Track 1",
      "data": [
        "09:00AM Writing Fast Tests Against Enterprise Rails 60min",
        "10:00AM Overdoing it in Python 45min",
        "10:45AM Lua for the Masses 30min",
        "11:15AM Ruby Errors from Mismatched Gem Versions 45min",
        "12:00PM Lunch",
        "01:00PM Ruby on Rails: Why We Should Move On 60min",
        "02:00PM Common Ruby Errors 45min",
        "02:45PM Pair Programming vs Noise 45min",
        "03:30PM Programming in the Boondocks of Seattle 30min",
        "04:00PM Ruby vs. Clojure for Back-End Development 30min",
        "04:30PM User Interface CSS in Rails Apps 30min",
        "05:00PM Networking Event"
      ]
    },
    {
      "title": "Track 2",
      "data": [
        "09:00AM Communicating Over Distance 60min",
```

```
                "10:00AM Rails Magic 60min",

                "11:00AM Woah 30min",

                "11:30AM Sit Down and Write 30min",

                "12:00PM Lunch",

                "01:00PM Accounting-Driven Development 45min",

                "01:45PM Clojure Ate Scala (on my project) 45min",

                "02:30PM A World Without HackerNews 30min",

                "03:00PM Ruby on Rails Legacy App Maintenance 60min",

                "04:00PM Rails for Python Developers lightning",

                "05:00PM Networking Event"

            ]

        }

        //Others tracks if necessary

    ]

}
```