Implemente uma API REST utilizando Node.js e Express. A API deve gerenciar os recursos:

Clientes (clientes)

Carros (carros)

Serviços (servicos)

Agendamentos (agendamentos)

A API deve simular o banco de dados usando listas em memória. Os dados inseridos ou atualizados devem ser armazenados na lista correspondente enquanto o servidor estiver em execução.

Requisitos:

**Endpoints para clientes (/clientes)**

Para as requisições que passam o código do cliente por parâmetro, caso o cliente não exista, deve retornar uma mensagem informado que o cliente não foi encontrado. Ex: {"message": "Cliente não encontrado"}

* GET /clientes
  + Descrição: Busca todos os clientes cadastrados.
  + Entrada:
  + Resposta: [{"id": 1, "nome": "João Silva", "telefone": "48999999999"}].
* POST /clientes
  + Descrição: Cadastra um novo cliente.
  + Entrada: {"nome": "Maria Souza", "telefone": "48988888888"}.
  + Resposta: {"mensagem": "Cliente cadastrado com sucesso"}
  + **Validações:**
    - nome:
      * Validação: Deve conter no mínimo 3 caracteres
      * Resposta: {"mensagem": "'nome' deve conter no mínimo 3 caracteres"}
      * Validação: Deve conter no máximo 100 caracteres.
      * Resposta: {"mensagem": "'nome' deve conter no máximo 100 caracteres"}
    - telefone:
      * Validação: Deve conter exatamente 11 dígitos.
      * Resposta: {"mensagem": "'telefone' deve conter exatamente 11 dígitos"}
      * Validação: Deve conter apenas números.
      * Resposta: {"mensagem": "'telefone' deve conter apenas números"}
* GET /clientes/:codigo
  + Descrição: Busca um cliente específico pelo código.
  + Entrada:
  + Resposta: {"id": 1, "nome": "João Silva", "telefone": "48999999999"}.
* PUT /clientes/:codigo
  + Descrição: Atualiza as informações de um cliente.
  + Entrada: {"id": 1, "nome": "João Carlos Silva", "telefone": "48977777777"}.
  + Resposta: {"mensagem": "Cliente atualizado com sucesso"}
  + **Validações:**
    - codigo:
      * Validação: Deve ser maior que 0
      * Resposta: {"mensagem": "'codigo' deve ser maior que 0"}
    - nome:
      * Validação: Deve conter no mínimo 3 caracteres
      * Resposta: {"mensagem": "'nome' deve conter no mínimo 3 caracteres"}
      * Validação: Deve conter no máximo 100 caracteres.
      * Resposta: {"mensagem": "'nome' deve conter no máximo 100 caracteres"}
    - telefone:
      * Validação: Deve conter exatamente 11 dígitos.
      * Resposta: {"mensagem": "'telefone' deve conter exatamente 11 dígitos"}
      * Validação: Deve conter apenas números.
      * Resposta: {"mensagem": "'telefone' deve conter apenas números"}
* DELETE /clientes/:codigo
  + Descrição: Remove um cliente pelo ID.
  + Entrada:
  + Resposta: {"mensagem": "Cliente removido com sucesso"}

**Endpoints para carros (/carros)**

Para as requisições que passam o código do carro por parâmetro, caso o carro não exista, deve retornar uma mensagem informado que o carro não foi encontrado. Ex: {"message": "Carro não encontrado"}

* GET /carros
  + Descrição: Busca todos os carros cadastrados.
  + Entrada:
  + Resposta: [{"id": 1, "marca": "Toyota", "modelo": "Corolla", "tamanho": "SEDAN", "id\_cliente": 1}].
* POST /carros
  + Descrição: Cadastra um novo carro.
  + Entrada: {"marca": "Toyota", "modelo": "Corolla", "tamanho": "SEDAN", "id\_cliente": 1}.
  + Resposta: {"mensagem": "Carro cadastrado com sucesso"}.
  + **Validações:**
    - marca:
      * Validação: Deve conter no mínimo 3 caracteres.
      * Resposta: {"mensagem": "'marca' deve conter mínimo 3 caracteres"}
      * Validação: Deve conter no máximo 50 caracteres.
      * Resposta: {"mensagem": "'marca' deve conter no máximo 50 caracteres"}
    - modelo:
      * Validação: Deve conter no mínimo 2 caracteres.
      * Resposta: {"mensagem": "'modelo' deve conter no mínimo 2 caracteres"}
      * Validação: Deve conter no máximo 50 caracteres.
      * Resposta: {"mensagem": "'modelo' deve conter no máximo 50 caracteres"}
    - tamanho:
      * Validação: Deve ser um dos valores: "HATCH", "SEDAN", "SUV", "PICAPE".
      * Exemplo de erro: {"mensagem": "'tamanho' deve ser HATCH, SEDAN, SUV ou PICAPE"}
    - id\_cliente:
      * Validação: Deve ser um cliente existente.
      * Resposta: {"mensagem": "'id\_cliente' não corresponde a um cliente cadastrado"}
* GET /carros/:codigo
  + Descrição: Busca um carro específico pelo código.
  + Entrada:
  + Resposta: {"id": 1, "marca": "Toyota", "modelo": "Corolla", "tamanho": "SEDAN", "id\_cliente": 1}.
* PUT /carros/:codigo
  + Descrição: Atualizar as informações de um carro.
  + Entrada: {"id": 1, "marca": "Toyota", "modelo": "Corolla", "tamanho": "HATCH", "id\_cliente": 1}.
  + Resposta: {"mensagem": "Carro Atualizado com sucesso"}.
  + **Validações:**
    - codigo:
      * Validação: Deve ser maior que 0
      * Resposta: {"mensagem": "'codigo' deve ser maior que 0"}
    - marca:
      * Validação: Deve conter no mínimo 3 caracteres.
      * Resposta: {"mensagem": "'marca' deve conter mínimo 3 caracteres"}
      * Validação: Deve conter no máximo 50 caracteres.
      * Resposta: {"mensagem": "'marca' deve conter no máximo 50 caracteres"}
    - modelo:
      * Validação: Deve conter no mínimo 2 caracteres.
      * Resposta: {"mensagem": "'modelo' deve conter no mínimo 2 caracteres"}
      * Validação: Deve conter no máximo 50 caracteres.
      * Resposta: {"mensagem": "'modelo' deve conter no máximo 50 caracteres"}
    - tamanho:
      * Validação: Deve ser um dos valores: "HATCH", "SEDAN", "SUV", "PICAPE".
      * Exemplo de erro: {"mensagem": "'tamanho' deve ser HATCH, SEDAN, SUV ou PICAPE"}
    - id\_cliente:
      * Validação: Deve ser um cliente existente.
      * Resposta: {"mensagem": "'id\_cliente' não corresponde a um cliente cadastrado"}
* DELETE /carros/:codigo
  + Descrição: Remove um carro pelo ID.
  + Entrada:
  + Resposta: {"mensagem": "Carro removido com sucesso"}

**Endpoints para serviços (/servicos)**

Para as requisições que passam o código do serviço por parâmetro, caso o serviço não exista, deve retornar uma mensagem informado que o serviço não foi encontrado. Ex: {"message": "Serviço não encontrado"}

* GET /servicos
  + Descrição: Busca todos os serviços cadastrados.
  + Entrada:
  + Resposta: [{"id": 1, "descricao": "Lavagem completa", "valores": [{"tamanho": "HATCH, "valor": "50.8}]}].
* POST /servicos
  + Descrição: Cadastra um novo serviço.
  + Entrada: {"descricao": "Lavagem completa", "valores": [{"tamanho": "HATCH, "valor": "50.8}]}.
  + Resposta: {"mensagem": "Serviço cadastrado com sucesso"}.
  + **Validações:**
    - descricao:
      * Validação: Deve conter no mínimo 5 caracteres.
      * Resposta: {"mensagem": "'descricao' deve conter no mínimo 5 caracteres"}
      * Validação: Deve conter no máximo 100 caracteres.
      * Resposta: {"mensagem": "'descricao' deve conter no máximo 100 caracteres"}
    - valores:
      * Validação: Deve conter valores iguais ou maiores que 0 para cada tamanho de carro.
      * Resposta: {"mensagem": "O valor para 'HATCH' deve ser igual ou maior que 0"}
* GET /servicos/:codigo
  + Descrição: Busca um serviço específico pelo código.
  + Entrada:
  + Resposta: {"id": 1, "descricao": "Lavagem completa", "valores": [{"tamanho": "HATCH, "valor": "50.8}]}.
* PUT /servicos:codigo
  + Descrição: Atualizar as informações de um serviço.
  + Entrada: {"id": 1, "descricao": "Lavagem completa", "valores": [{"tamanho": "HATCH, "valor": "50.8}]}.
  + Resposta: {"mensagem": "Serviço atualizado com sucesso"}.
  + **Validações:**
    - codigo:
      * Validação: Deve ser maior que 0
      * Resposta: {"mensagem": "'codigo' deve ser maior que 0"}
    - descricao:
      * Validação: Deve conter no mínimo 5 caracteres.
      * Resposta: {"mensagem": "'descricao' deve conter no mínimo 5 caracteres"}
      * Validação: Deve conter no máximo 100 caracteres.
      * Resposta: {"mensagem": "'descricao' deve conter no máximo 100 caracteres"}
    - valores:
      * Validação: Deve conter valores iguais ou maiores que 0 para cada tamanho de carro.
      * Resposta: {"mensagem": "O valor para 'HATCH' deve ser igual ou maior que 0"}
* DELETE /servicos/:codigo
  + Descrição: Remove um serviço pelo ID.
  + Entrada:
  + Resposta: {"mensagem": "Serviço removido com sucesso"}

**Endpoints para agendamentos (/agendamentos)**

Para as requisições que passam o código do agendamento por parâmetro, caso o agendamento não exista, deve retornar uma mensagem informado que o agendamento não foi encontrado. Ex: {"message": "Agendamento não encontrado"}

* GET /agendamentos
  + Descrição: Busca todos os agendamentos cadastrados.
  + Entrada:
  + Resposta: [{"id\_carro": 1, "id\_servico": 1, "data\_hora": "2024-10-25T14:00:00"}]
* POST /agendamentos
  + Descrição: Cadastra um novo agendamento.
  + Entrada: {"id\_carro": 1, "id\_servico": 1, "data\_hora": "2024-10-25T14:00:00"}.
  + Resposta: {"mensagem": "Agendamento cadastrado com sucesso"}.
  + **Validações:**
    - id\_carro:
      * Validação: Deve ser um carro existente.
      * Resposta: {"mensagem": "'id\_carro' não corresponde a um carro cadastrado"}
    - id\_servico:
      * Validação: Deve ser um serviço existente.
      * Resposta: {"mensagem": "'id\_servico' não corresponde a um serviço cadastrado"}
      * data\_hora: Deve ser informado uma data e hora.  
        Resposta: {"mensagem": "'data\_hora' deve ser informado"}

Regras e Dicas:

Utilize arrays para armazenar os clientes, carros, serviços e agendamentos. Por exemplo:  
let clientes = [];  
let carros = [];  
let servicos = [];  
let agendamentos = [];  
Cada novo recurso inserido deve receber um id único gerado automaticamente.  
Utilize a biblioteca Express para gerenciar as rotas e lidar com as requisições.  
Valide os dados recebidos no corpo das requisições.  
Retorne mensagens claras e códigos de status HTTP apropriados para cada operação.  
**No final do arquivo da API, deve constar o seguinte trecho de código: *module.exports = suaVariavelApi.***No lugar da variável ***suaVariavelApi***, coloque a sua variável responsável por armazenar o objeto do servidor. Geralmente após a função ***.listen();***

Entrega:

Deve ser entregue o link do github e o código deve ser funcional e bem estruturado, e todos os endpoints devem estar testados e funcionando corretamente. No repositório deve constar o arquivo responsável por rodar a API e o arquivo package.json.

Avaliação:

Será avaliado funcionalidade da API como um todo.  
Está respondendo com os códigos de estatus correto: 200, 404...  
Está retornando mensagem avisando que o recurso em questão não existe, caso necessário.