Para iniciarmos siga a parte 4, e para validarmos as entradas invalida no react é algo bem simples. Nós iremos criar um campo para rastrear e atualizar os erros de validação logo abaixo do de mudança de mensagem (fica abaixo de Function app(), e novamente usando useState para poder alterar o estado do componente

```
const [errors, setErrors] = useState({});
```

Logo abaixo de handle change iremos criar a função setErrors para atualizar o estado do erro, se ainda esta errado os campos preenchidos ou não

Usaremos ...prevErrors para copiar os estados dos erros anteriores para não perde-los caso aconteça algum problema, usaremos [name]: undefined que define um novo erro ou limpa um erro existente associado ao campo de entrada especificado pelo name. Isso é feito atribuindo undefined como o valor para a chave name no objeto de erros. Dizendo que não tem erro no campo especificado, logo ficara

```
setErrors(prevErrors => ({
    ...prevErrors,
    [name]: undefined
  }));
};
```

Antes de enviar a requisição para a API com o formulário em forma de objeto, logo abaixo de setErrors criaremos a função para verificar se tem algum erro ou não no preenchimento do questionário conforme a nossa validação, usando uma função bem simples que verifica se o objeto de erro é maior que 0 indicando que existe um erro, caso tenha chamara a nossa função setErrors mandando o objeto para ser exibido na tela e logo corrigido.

```
const validationErrors = validate(formData);
  if (Object.keys(validationErrors).length > 0) {
    setErrors(validationErrors);
    return;
  }
```

Abaixo da requisição na API criaremos a função para verificar retornar o erro presente caso exista e sua condição e mensagem para correção, coisas básicas como verificar se o texto tem menos de 2 caracteres ou mais que 255, logo ficara

```
const validate = (data) => {
  let errors = {};

  if (!data.texto || data.texto.length < 2 || data.texto.length > 255) {
    errors.texto = "Texto é obrigatório e deve ter entre 2 e 255 caracteres.";
  }

  if (data.inteiro === null || data.inteiro <= 0 || data.inteiro >= 1000) {
    errors.inteiro = "Inteiro é obrigatório e deve ser maior que 0 e menor que 1000.";
  }

  if (data.booleano !== true && data.booleano !== false) {
    errors.booleano = "Booleano é obrigatório.";
  }

  if (!data.opcaoSelect) {
    errors.opcaoSelect = "Selecione uma opção do dropdown.";
  }

  if (!data.opcaoRadio) {
    errors.opcaoRadio = "Selecione uma opção nos botões de rádio.";
  }

  return errors;
};
```

Agora por fim mas sim uma modificação, abaixo de todos os campos separados do nosso formulário criado na parte 4, chamaremos a função validate passando como parâmetro o nome campo a ser validado e o erro será exibido em um span caso o resultado seja true (contenha erro), caso contrario não será exibido nada, logo abaixo do campo coloque a seguinte formula

{errors.texto && {errors.texto}}

Não mudaria nada para os outros campos do questionário, somente o nome, por exemplo para o boolean ficaria :

{errors.booleano && {errors.booleano}}

Logo ficara assim o nosso questionario

```
return (
  <div>
    <form onSubmit={handleSubmit}>
      <label>
        Texto:
        <input type="text" name="texto" value={formData.texto} onChange={handleChange} />
        {errors.texto && <span>{errors.texto}</span>}
      </label>
      <br />
      <label>
        Inteiro:
        <input type="number" name="inteiro" value={formData.inteiro || ''} onChange={handleChange} />
        {errors.inteiro && <span>{errors.inteiro}</span>}
      </label>
      <br />
      <label>
        Booleano:
        <input type="checkbox" name="booleano" checked={formData.booleano} onChange={handleChange} />
        {errors.booleano && <span>{errors.booleano}</span>}
      </label>
      <br />
      <label>
        Opção Select:
        <select name="opcaoSelect" value={formData.opcaoSelect} onChange={handleChange}>      You, 5 hours ago • Uncommitted changes
          <option value="">Selecione...</option>
          <option value="opcao1">Torresmo</option>
          <option value="opcao2">Carne</option>
          <option value="opcao3">Feijao</option>
        </select>
        {errors.opcaoSelect && <span>{errors.opcaoSelect}</span>}
      </label>
      <br />
      <label>
```

```
        <br />
        <label>
          Opção Radio:
          <label>
            <input type="radio" name="opcaoRadio" value="opcaoA" checked={formData.opcaoRadio === "opcaoA"} onChange={handleChange} />
            Opção A
          </label>
          <label>
            <input type="radio" name="opcaoRadio" value="opcaoB" checked={formData.opcaoRadio === "opcaoB"} onChange={handleChange} />
            Opção B
          </label>
          {errors.opcaoRadio && <span>{errors.opcaoRadio}</span>}
        </label>
        <br />
        <button type="submit">Enviar</button>
      </form>
      {responseMessage && <p>{responseMessage}</p>}
    </div>
  );
}

export default App;
```

E para validar diretamente na API de c# é mais simples ainda, bem parecido com o conteúdo que presenciamos em algoritmo no primeiro período, siga os passos

1 – Criar uma variável para armazenar a contagem de erro para verificar se existe caso seja maior que 0 e mandar para uma função que criaremos mais pra frente, juntamente com o questionário como parâmetro

var validationErrors = Validate(formData);

Criar uma verificação usando if para verificar se existe algum erro e retornar para o usuário o erro caso ele exista, se não existir ficara abaixo desse if e retornara normal, o if será assim

```
if (validationErrors.Count > 0) {
  httpContext.Response.StatusCode = StatusCodes.Status400BadRequest;
  await httpContext.Response.WriteAsJsonAsync(validationErrors);
  return;
}
```

E logo abaixo retornara normal caso nao tenha nenhum erro

await httpContext.Response.WriteAsJsonAsync(formData);

Logo ficara assim

```
// Rota para receber dados do formulário
app.MapPost("/formulario", async (HttpContext httpContext) =>
{
    // Ler e desserializar dados do corpo da requisição
    var requestBody = await new System.IO.StreamReader(httpContext.Request.Body).ReadToEndAsync();
    var formData = JsonSerializer.Deserialize<FormData>(requestBody, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    // Validar os dados do formulário
    var validationErrors = Validate(formData);
    if (validationErrors.Count > 0)
    {
        // Se houver erros de validação, retornar uma resposta de erro com os detalhes dos erros
        httpContext.Response.StatusCode = StatusCodes.Status400BadRequest;
        await httpContext.Response.WriteAsJsonAsync(validationErrors);
        return;
    }

    // Se os dados forem válidos, retornar os dados recebidos
    await httpContext.Response.WriteAsJsonAsync(formData);
});
```

E logo abaixo criaremos a função para validar com a nossa logica e colocar os erros dentro de um array para depois olharmos o comprimento para identificar se tem erros e o conteúdo para identificar qual os erros para retornar ao usuário, ao invés de retornar o conteúdo

```
List<string> Validate(FormData formData)
{
   var errors = new List<string>();

   // Validar cada campo do formulário conforme necessário
   if (string.IsNullOrWhiteSpace(formData.Texto) || formData.Texto.Length < 2 || formData.Texto.Length
> 255)
   {
      errors.Add("Texto é obrigatório e deve ter entre 2 e 255 caracteres.");
   }

   if (formData.Inteiro <= 0 || formData.Inteiro >= 1000)
   {
      errors.Add("Inteiro é obrigatório e deve ser maior que 0 e menor que 1000.");
   }

   if (!formData.Booleano)
   {
      errors.Add("Booleano é obrigatório.");
   }

   if (string.IsNullOrWhiteSpace(formData.OpcaoSelect))
   {
      errors.Add("Selecione uma opção do dropdown.");
   }

   if (string.IsNullOrWhiteSpace(formData.OpcaoRadio))
   {
      errors.Add("Selecione uma opção nos botões de rádio.");
   }

   return errors;
}
```