sexta-feira, 22 de setembro de 2023 11:56

sempre_alerta_doc_socket_tempo_real_#1

Data Revisão Atual: 25/09/23
Data Revisão Anterior: n/a

◉ DEFINIÇÕES GERAIS

- ### Socket:

  Socket.IO é uma biblioteca que permite **baixa-latência**, **bidirecional** e **baseado em eventos** de comunicação em tempo real entre cliente e um servidor.

  Referência: https://socket.io/pt-br/docs/v4/

- ### Emitidor:

  Começa e envia um evento geralmente com dados disparado por alguma ação do usuário ou de forma programática. É uma função que começa com 'emit'.

- ### Escutador:

  Detecta e recebe um evento vindo de um emitidor. É um método que começa com 'listen'.

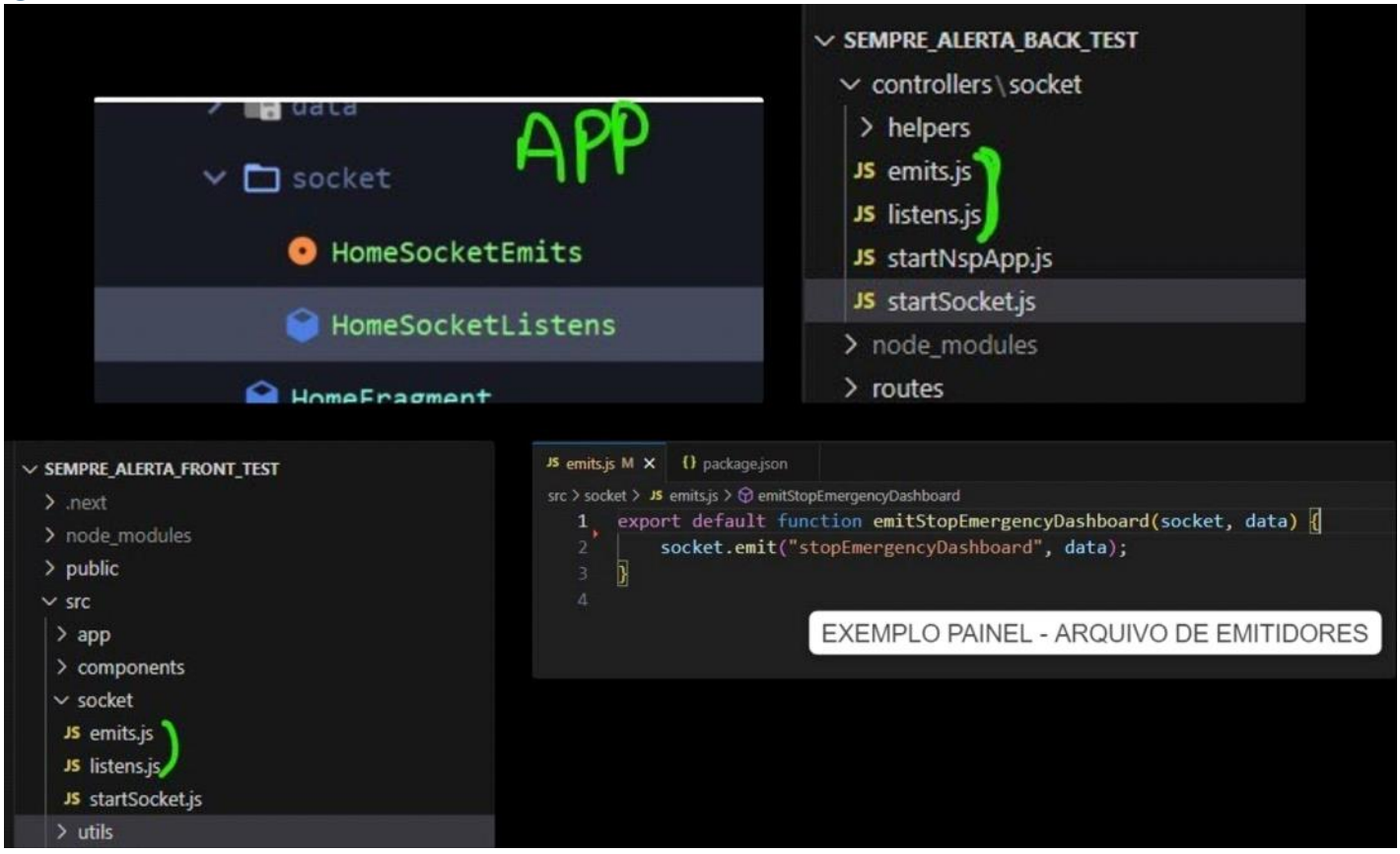
- ### Evento:

  É a nome que identifica cada ação a ser executa. Esse nome, por convenção, é composto por 'ação' + 'origem/destino'

- ### Namespace (abreviado como Nsp):

  No socket, é o canal de comunicação que permite dividir a lógica na mesma conexão (também chamado de 'multiplexing'). Atualmente só tem um único namespace "App" que agrupa todos os agentes do socket que trabalham no app. Mas pode-se criar namespace para diferentes telas como uma namespace para tela configuração, outro para tela principal, etc.

◉ ESTRUTURA GERAL SOCKET NOS AMBIENTES DO PROJETO



- ### Ambientes:

- Para facilitar o desenvolvimento, todos os ambientes do projeto seguem o mesma estrutura de pastas e arquivos para gerenciar o socket. São compostos por 3 ambientes:

    a) App: possui emitidores e escutadores do backend;
    b) Painel: mesmo coisa do app;
    c) Backend: Intermedia todos os eventos entre app e painel.

    Analogia: No ping-pong, o backend seria a mesa e as raquetes seriam de um lado o app e de outro o painel. A bolinha seria os eventos.

▪ Linguagens de programação:

- App: Kotlin
- Painel: Javascript (pode ser alterado)
- Backend: NodeJs (pode ser alterado)

▪ Pastas:

- Todos os emitidores e escutadores se encontram em arquivos separados dentro da pasta 'socket'.

- A pasta de 'emits' possui todos os emitidores podendo agrupar todos os métodos ou componentizar de acordo com a tela ou aba do ambiente do projeto.

- A pasta de 'listens' possui todos os escutadores.

◉ MAPA EVENTOS SOCKET SEMPRE ALERTA

Lista de todos os eventos que interagem nos ambientes do projeto.

1) startEmergency (emitido App) - quando usuário clica no botão SOS
2) stopEmergency (emitido Painel) - quando usuário clica no botão "Desativar Alerta"

APP	BACKEND	PAINEL
startEmergencyApp (emit) ->	-> startEmergencyApp (event) -> startEmergencyDashboard (emit)	-> startEmergencyDashboard (event)
stopEmergencyApp (event) <-	<- stopEmergencyApp (emit) <- stopEmergencyDashboard (event)	<- stopEmergencyDashboard (emit)

Legenda de setas

- As setas indicam onde começa um evento em tempo real
(->) Seta direta indica que o emitidor começou no APP
(<-) Seta esquerda indica que o emitidor começou no PAINEL
(<-b->) Seta bilateral indica que o emitidor começou no BACKEND e emite todos os ambientes.
Exemplo: checagem de desconexão no servidor e reconexão com todas as salas (ainda não necessário implementar porque o método do socket de reconexão foi otimizado)

◉ CONVENÇÃO DE NOMES PARA MÉTODOS

- A convenção ajuda na criação dos nomes dos métodos. Ajudando no desenvolvimento ao identificar rapidamente a ação de cada tipo de agente socket dentro do projeto.

▪ Nomes para os métodos (emits e listens):

- Nomes seguem o estilo 'camel case'.
- Estrutura do nome:

    TIPO AGENTE SOCKET (emitidor ou escutador) + EVENTO (nome ação) + ORIGEM/DESTINO

    a) Tipo agente socket: use-se no começo dos métodos: 'emit' para emitidores e 'listen' para escutadores.

    b) Nome do evento ou ação

    c) Origem/Destino
    Os nomes do eventos precisam da origem ou destino para não haver conflito de qual ambiente será disparado o evento, principalmente se originar do backend e precisar enviar um evento somente para o app e não para o painel, por exemplo.

    DIFERENÇA DE ORIGEM E DESTINO
    - Se emite ou escuta um evento do app ou painel: É a ORIGEM.
    Exemplo: No app, o emitidor chamado 'emitStartEmergencyApp' emite o evento 'startEmergencyApp' onde o nome App refere-se a origem, do próprio ambiente onde foi declarado, seja app ou painel.

    - Se emitir um evento do backend, o nome refere-se ao DESTINO do evento.
    Exemplo: No backend, o emitidor chamado 'emitStartEmergencyDashboard' emite o evento 'startEmergencyDashboard' onde nome Dashboard refere-se ao destino, de onde um evento

com mesmo nome receberá o evento.

- Nomes para os eventos:
  EVENTO (nome ação) + ORIGEM/DESTINO

  Não inclui o tipo de agente socket (emit/listen).

◉ ORIGEM NO BACKEND

Como mostrado, o backend detecta a origem dos eventos emitidos, seja do painel ou app. É sempre enviado via query string na inicialização do socket.

Assim é possível gerenciar ações e fazer lógicas de acordo com determinado ambiente.

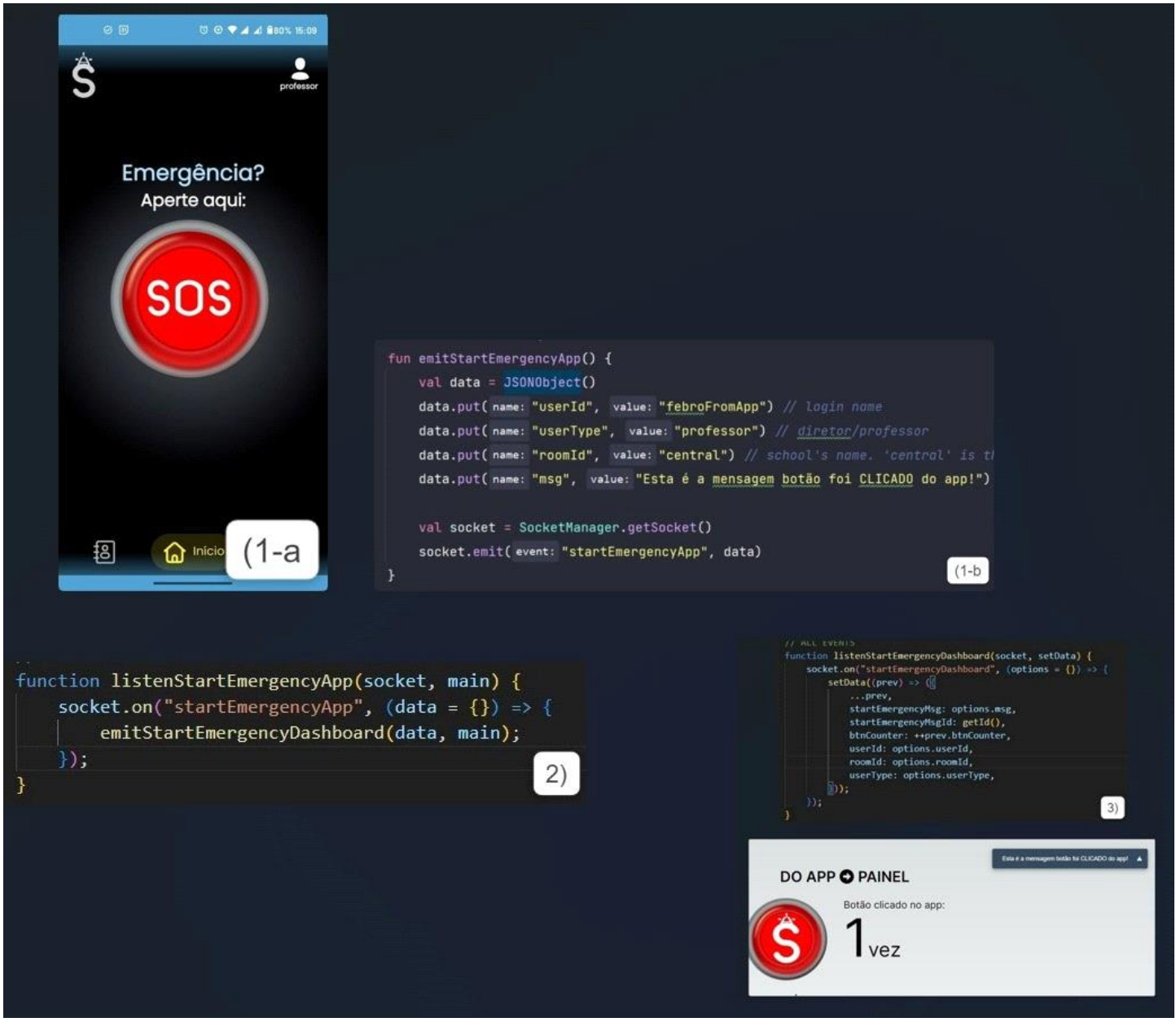
```
nspApp.on("connection", async (socket) => {
    console.log("starting socket successfully!");

    const { origin } = socket.query;
    if (origin === "app") console.log("new connection from APP");
    else
        console.log(
            `new connection from ${origin && origin.toUpperCase()}`
        );
```

```
val options = IO.Options.builder()
    .setForceNew(false)
    .setReconnection(true)
    .setReconnectionAttempts(Int.MAX_VALUE) //
    .setReconnectionDelay(1000) // The initial
    .setReconnectionDelayMax(5000) // The maxim
    .setRandomizationFactor(0.5) // The randomi
    .setTimeout(20000) // The timeout in millis
    .setPath("/socket.io/")
    .setQuery("origin=app")
    .setTransports(arrayOf(WebSocket.NAME, Poll
```

```
    reconnectionDelayMax: 5000, // The ma
    randomizationFactor: 0.5, //  n1
    timeout: 2000,
    autoConnect: true,
    path: "/socket.io",
    query: { origin: "dashboard" },
    transports: ["websocket", "polling"],
```

◉ EXEMPLO DE COMPORTAMENTO EVENTO APP -> PAINEL



1-a) APP - Usuário clica botão SOS.

1-b) APP - Ao clicar, o emitidor "emitStartEmergencyApp" dispara o evento "startEmergencyApp" com dados incluindo: userId, userType, roomId, msg.

2) BACKEND TESTE - O escutador "startEmergencyApp" recebe os dados em tempo real e emite o evento "startEmergencyDashboard" onde o ambiente destino é painel teste

3) PAINEL TESTE - recebe os dados de "startEmergencyDashboard" do BACKEND TESTE e executa ações no painel como mostrar mensagem e registrar quantidades de ação.

◉ COMPORTAMENTO DAS SALAS

▪ Definição

Sala representa cada grupo onde será enviado determinado evento com dados. Por exemplo, um evento precisa ser disparado apenas para um grupo de pessoas específicas, não para todo mundo. Aqui entra a função de cada sala.

▪ Sala em cada ambiente e cadastro de salas

- Quem acessa painel, participa de todas as salas cadastradas. Precisa guardar todas as listas de salas no banco de dados e fazer o usuário participar dessas salas na inicialização.
- Quem acessa app, participa de sua sala identificada no login.
- No backend, só é gerenciado as salas dos demais ambientes.

▪ Quando o usuário participa

Cada usuário participa da sala do socket assim que entra no app ou no painel, na tela principal.

◉ ESTRATÉGIAS DE REATIVAÇÃO DO SOCKET DESCONECTADO/INATIVO

O socket pode deixar de funcionar ou desconectar quando o usuário trocar de aba no painel ou, no app, quando o app fica aberto no background ou parado por muito tempo. Mas há estratégias utilizadas para contornar tais problemas recorrentes em software de tempo real.

Atualização: uma melhoria foi feita a partir da versão 1.17.0 ao reduzir o tempo de reconexão de 20 segundos para 2 segundos, fazendo o socket reconectar mais rápida.

▪ No app

O socket é reativado no app a cada interação com a tela inicial mesmo se o app aberto. Essa reativação acontece dentro do ciclo do onStart no Android.

▪ No painel

Quando o usuário deixa a tela do painel aberta, mas troca de abas do navegador, é provável que o aba fique inativo e o socket seja desativado. Para resolver isso, é usado o evento onFocus para reinicializar o socket e a tela continuar rastreando eventos em tempo real. (ainda não implementado)

◉ ÚLTIMA ATUALIZAÇÃO CÓDIGO FONTE:

APP (sempre_alerta_v1.17.0)
https://github.com/OLSdesenvolvimento/SempreAlerta

BACKEND TESTE (test_sempre_alerta_back_v0.1.3)
https://github.com/luisFebro/test_sempre_alerta_back

FRONTEND TESTE (test_sempre_alerta_front_v0.1.2)
https://github.com/luisFebro/test_sempre_alerta_front