

# Licenciatura em Engenharia Informática

SISTEMA DE NOTIFICAÇÃO DE AVISOS SOBRE ALTERAÇÃO NOS  
HORÁRIOS DOS COMBOIOS

Grupo 9

Axel Guimarães - 8180657

José Pereira - 8200254

Diogo Silva - 8200573

SISTEMAS DISTRIBUÍDOS

# Conteúdo

Introdução .....	2
Observações técnicas .....	3
Descrição dos componentes implementados.....	4
server .....	4
<i>Server</i> .....	4
<i>Protocol</i> .....	4
<i>JsonFileHelper</i> .....	4
<i>ClientHandler</i> .....	4
<i>ArrayListSync</i> .....	4
userInterface .....	4
<i>Client</i> .....	4
<i>GUI/InitialFrame</i> .....	5
<i>GUI/LocalNodeMenuFrame</i> .....	6
<i>GUI/PassengerMenuFrame</i> .....	6
utils.....	7
<i>Request</i> .....	7
<i>Response</i> .....	8
<i>RequestType</i> .....	9
<i>ResponseStatus</i> .....	9
Estrutura do sistema e fluxos de atividades .....	10
Decisões tomadas .....	15
Conclusão .....	16
Referências .....	17

## Introdução

No âmbito da avaliação prática da unidade curricular de Sistemas Distribuídos, foi proposta a elaboração de um sistema de notificação de avisos sobre alteração nos horários dos comboios. Este projeto visa a simulação de um sistema de notificação de avisos sobre a alteração nos horários dos comboios. O objetivo é o sistema notificar os intervenientes do sistema (rede ferroviária) da existência de alterações nos horários dos comboios enviando avisos mediante um evento, a incidência do mesmo e ficando à espera de receber alguma resposta.

Este projeto foi desenvolvido em linguagem de programação *Java* e compilador *Gradle*, com recurso a *multithreading*, de modo a um funcionamento ordenado de recursos TCP e UDP para a comunicação entre partes.

## Observações técnicas

- Este projeto foi desenvolvido na versão SDK 13.0.14 com auxílio do IntelliJ IDE;
- Toda a interface gráfica *Java Swing* foi construída e executada com recurso ao IntelliJ IDEA. Em caso de erro ao iniciar as interfaces gráficas, verificar as definições de *build* e *run* em FILE -> Settings -> Build, Execution, Deployment -> Build tools -> Gradle -> Build and run using: IntelliJ IDEA; Run tests using: IntelliJ IDEA. Este problema surge devido a uma recente atualização do *Gradle*, que deixou de instanciar automaticamente todas as variáveis relativas ao *Java Swing*. (Para mais informações: <https://stackoverflow.com/questions/55844177/intellij-swing-gui-not-compiling-because-of-gradle>).

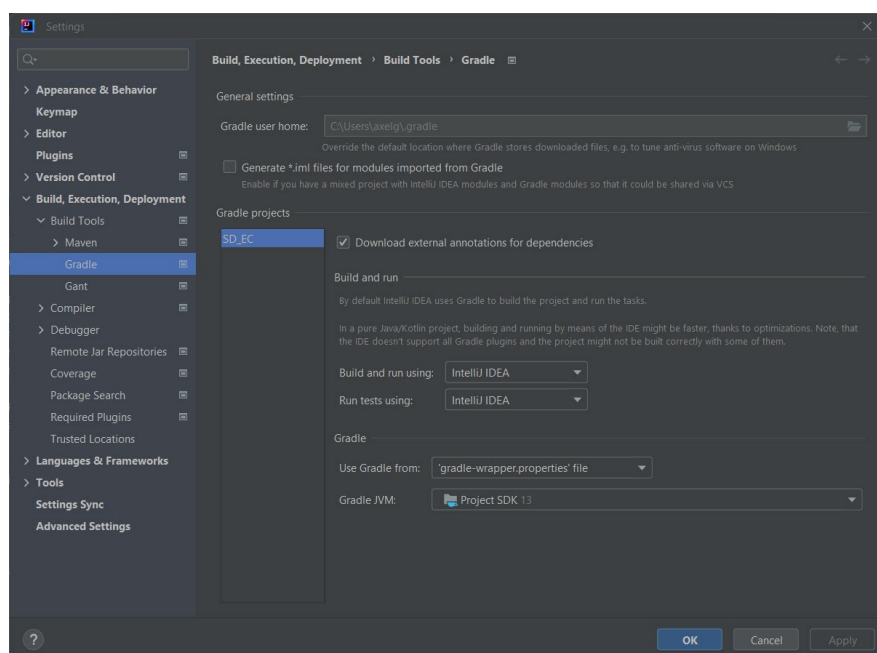


Figura 1 - Definições de build e run

## Descrição dos componentes implementados

Nos subtópicos que se seguem, será dada uma breve descrição dos componentes desenvolvidos no projeto.

### SERVER

Os seguintes componentes estão incluídos no *package* Server:

#### Server

Esta classe recebe as conexões e pedidos dos clientes, criando uma *thread* para cada conexão de cliente.

#### Protocol

Esta classe define o protocolo de comunicação entre o cliente e o servidor. Esta classe trata de analisar o tipo de pedido que o cliente manda para o servidor, e redireciona-o para o método específico ao pedido. Como será explicado mais à frente, todos os pedidos e respostas são trocados em formato JSON. A conversão destes pedidos e respostas para formato JSON é feita com auxílio da biblioteca Gson.

#### JsonFileHelper

Esta classe é responsável pela leitura/escrita de/em ficheiros JSON que armazenam os dados relativos a passageiros e gestores locais.

#### ClientHandler

Esta classe cuida de cada instância de cliente criada, e é responsável por receber e enviar todas as mensagens relativas à respetiva instância de cliente.

#### ArrayListSync

Esta classe faz o papel de uma ArrayList, mas com a particularidade de todos os seus métodos serem *synchronized*, tornando-os *thread-safe*.

### USERINTERFACE

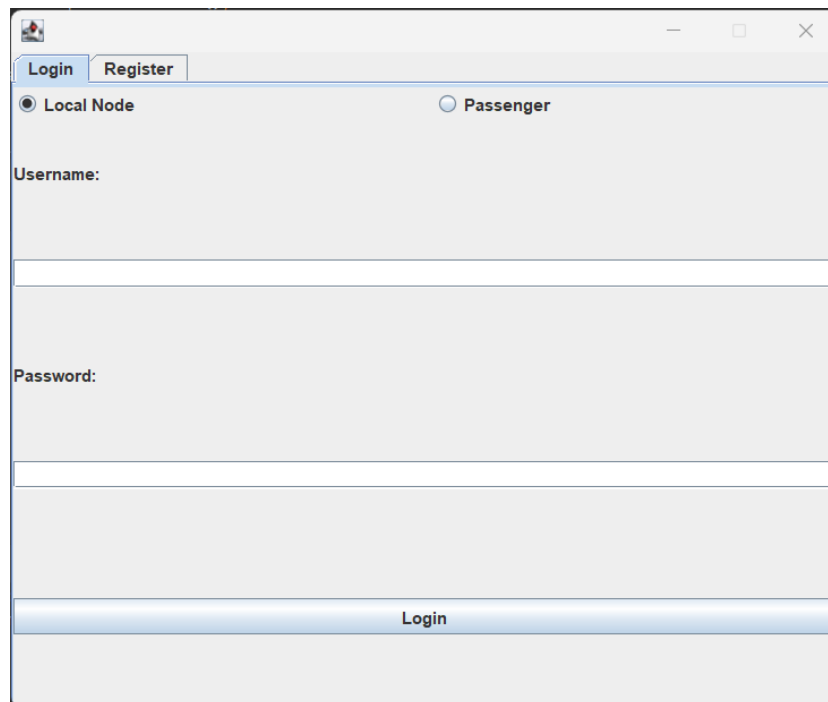
Os seguintes componentes estão incluídos no *package* userInterface:

#### Client

Esta classe representa cada cliente que se pretende conectar ao servidor. Ao se iniciar uma instância de Client, é criado um socket de ligação ao servidor, com as portas 2048 para TCP e 4446 para UDP (Multicast). São iniciados dois *threads* para receber mensagens normais e mensagens *multicast* respetivamente. Além disso, é disponibilizado um método que lança uma *thread* de envio de mensagens, que pode ser chamado quando necessário.

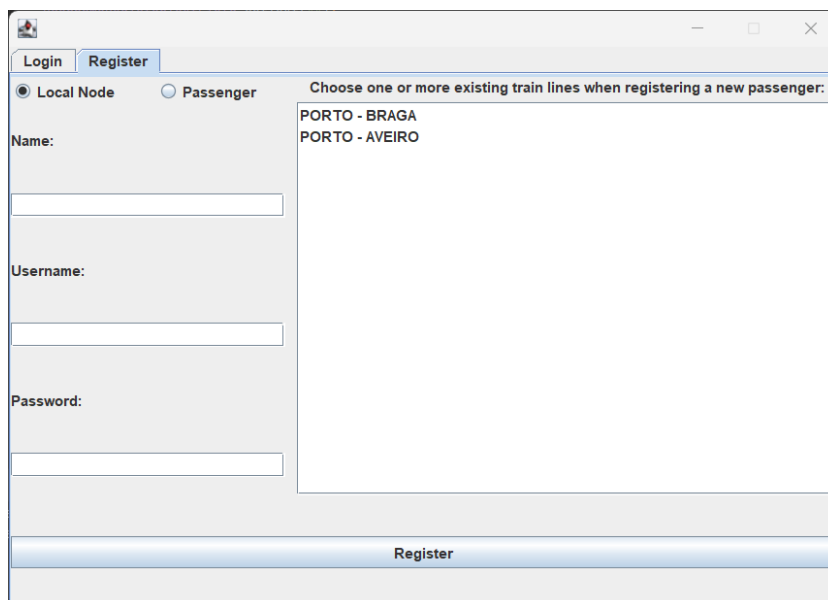
## GUI/InitialFrame

Esta classe representa a interface gráfica inicial apresentada a cada cliente no ato de *login* ou de registo. Nesta janela é possível alternar entre *login*/registo através do respetivo separador, e em cada um, é possível selecionar se pretende fazer *login*/registo de um gestor local ou de um passageiro.



The screenshot shows a window titled "Login" with a tabbed interface. The "Login" tab is selected. Below the tabs, there are two radio buttons: "Local Node" (selected) and "Passenger". Below these, there are two text input fields labeled "Username:" and "Password:". At the bottom, there is a "Login" button.

*Figura 2 - Janela inicial no separador Login*

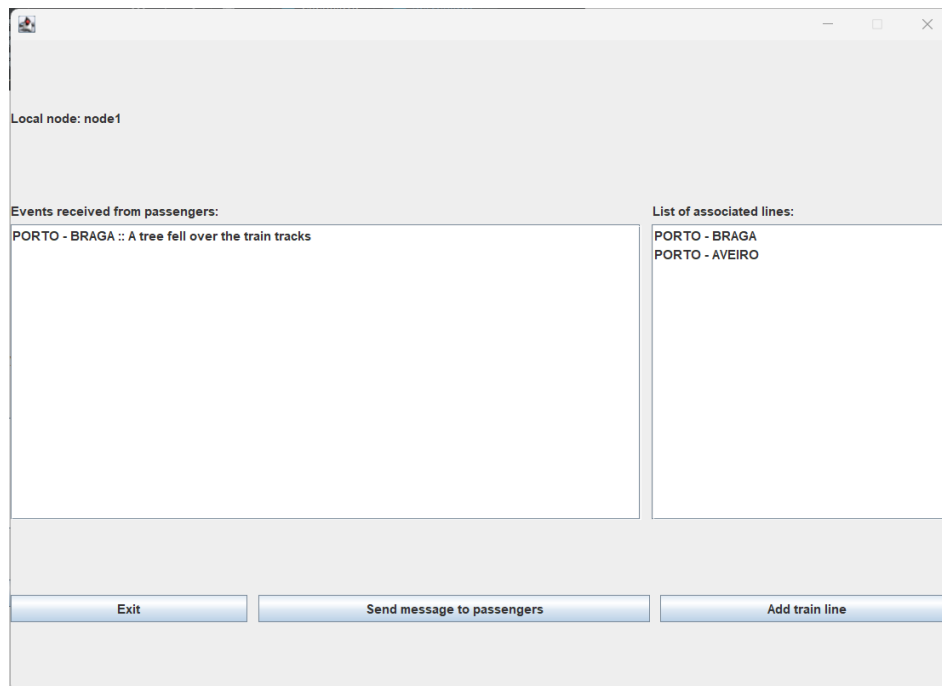


The screenshot shows a window titled "Register" with a tabbed interface. The "Register" tab is selected. Below the tabs, there are two radio buttons: "Local Node" (selected) and "Passenger". Below these, there are three text input fields labeled "Name:", "Username:", and "Password:". To the right of these fields, there is a list box titled "Choose one or more existing train lines when registering a new passenger:". The list box contains two items: "PORTO - BRAGA" and "PORTO - AVEIRO". At the bottom, there is a "Register" button.

*Figura 3 - Janela inicial no separador Register*

### GUI/LocalNodeMenuFrame

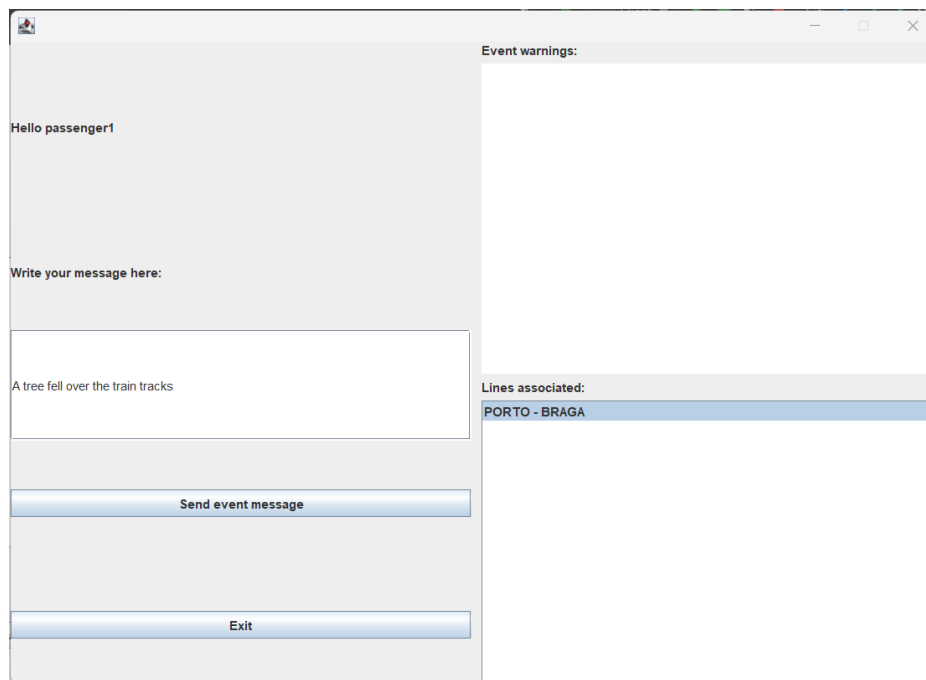
Esta classe representa a interface gráfica apresentada após o *login* de cada gestor local. Aqui é possível adicionar (criar) linhas ferroviárias ao gestor local, visualizar as mensagens recebidas por passageiros associados às linhas existentes no gestor local, e enviar notificações referentes a uma linha ferroviária, a todos os passageiros associados a essa linha.



*Figura 4 - Janela do menu de Gestor Local*

### GUI/PassengerMenuFrame

Esta classe representa a interface gráfica após o *login* de cada passageiro. Aqui é possível visualizar todas as linhas associadas no ato de registo do passageiro, visualizar as notificações enviadas por gestores locais aos quais as linhas associadas pertencem, e enviar mensagens de evento ao gestor local de uma respetiva linha ferroviária.



*Figura 5 - Janela de menu de Passageiro*

## UTILS

O package *utils* contém todas as classes que esquematizam os vários componentes usados no projeto. Neste subtópico serão apenas mencionadas as mais relevantes, dado que as restantes servem apenas de auxílio para a estruturação dos dados quando é necessário haver conversão de JSON-Objeto e Objeto-JSON.

### Request

Como mencionado acima, a classe *Protocol* faz uso de *requests* e *responses* na troca de mensagens entre o servidor e os clientes. Todos os pedidos feitos ao servidor utilizam a classe *Request*. Esta classe é genérica, podendo representar qualquer tipo de pedido necessário, de acordo com os tipos de pedidos existentes, definidos na classe *RequestType*. A porção de código abaixo representa a classe *Request*:

```
public class Request<T> {  
    private RequestType type;  
    private T data;  
  
    public Request(RequestType type, T data) {  
        this.type = type;  
        this.data = data;  
    }  
}
```

*Figura 6 - Classe Request*



Um exemplo de Request para o *login* de um gestor local é o seguinte:

```
{
  "type": "LOCAL_NODE_LOGIN",
  "data": {
    "username": "node1",
    "password": "$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$5uT
uHzGrQDoRuNwNUztsZ/6VQE0kNlS5sp4KCUS3pHk"
  }
}
```

*Figura 7 - Request Local Node Login*

## Response

Assim como a classe Request, a classe Response é usada em todas as respostas dadas pelo servidor. Esta classe também é genérica, podendo representar qualquer tipo de resposta necessária, de acordo com os tipos de respostas existentes, definidos na classe ResponseStatus. A porção de código abaixo representa a classe Response:

```
public class Response<T> {
    public ResponseStatus status;
    public RequestType type;
    public String message;
    public T data;

    public Response(ResponseStatus status, RequestType type, String
message, T data) {
        this.status = status;
        this.type = type;
        this.message = message;
        this.data = data;
    }
}
```

*Figura 8 – Classe Response*

Um exemplo de Response para o *login* de um gestor local é o seguinte:

```

{
  "status": "OK",
  "type": "LOCAL_NODE_LOGIN",
  "message": "Login successfully!",
  "data": {
    "localNode": {
      "name": "node1",
      "username": "node1",
      "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$5uTuHzGrQDoRuNwNUzts
Z/6VQE0kN1S5sp4KCUS3pHk",
      "passengers": [
        {
          "name": "passenger1",
          "username": "passenger1",
          "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$izhQY0YZkdm2DgXvtXJ+
Z1nnNAd2iKPKaiv0dbK7zd0",
          "addedTrainLines": [{ "beginning": "porto", "end": "braga" }]
        },
        {
          "name": "passenger2",
          "username": "passenger2",
          "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$QC6jvWT1EamM+Pfu+AZ1
3ZNuH/QVB1DV2b54VXiaWLQ",
          "addedTrainLines": [
            { "beginning": "porto", "end": "braga" },
            { "beginning": "porto", "end": "aveiro" }
          ]
        }
      ],
      "trainLines": [
        { "beginning": "porto", "end": "braga" },
        { "beginning": "porto", "end": "aveiro" }
      ]
    },
    "listIpsToJoin": ["224.0.0.2", "224.0.2.0"]
  }
}

```

*Figura 9 - Response Local Node Login*

### RequestType

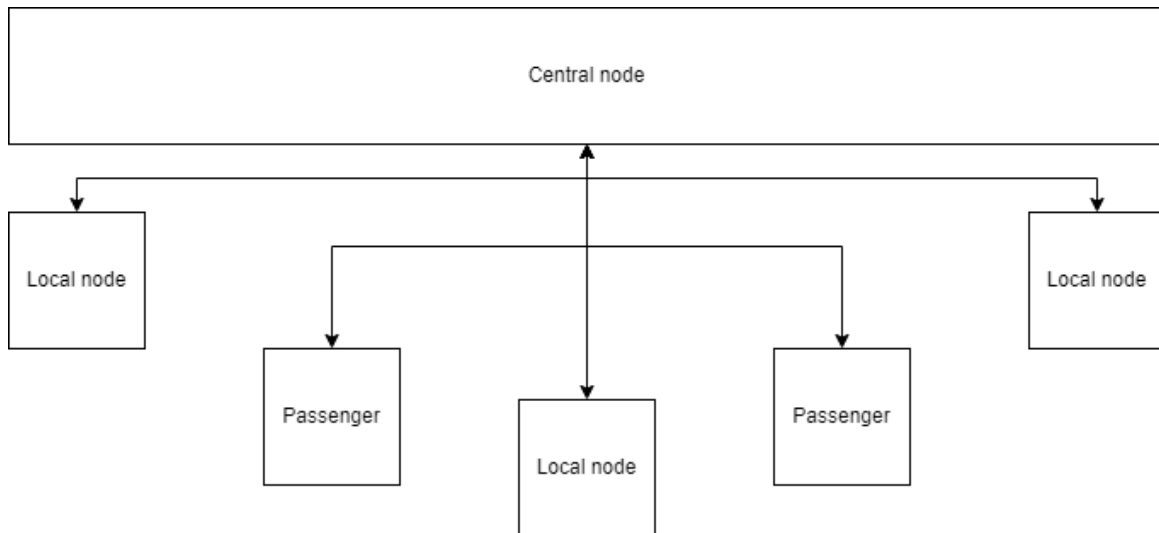
Esta classe é uma enumeração que define os tipos de pedidos que podem ser feitos ao servidor.

### ResponseStatus

Esta classe é uma enumeração que define os tipos de respostas que podem ser dadas pelo servidor.

## Estrutura do sistema e fluxos de atividades

Dado a este sistema ser em contexto servidor-cliente, bem como este projeto ter sido desenvolvido localmente, não foi possível estabelecer ligações diretas entre passageiros e gestores locais, pelo que se decidiu em estabelecer todas as ligações de passageiros e gestores locais ao gestor central. Sendo assim, todas as mensagens trocadas entre gestores locais e passageiros passam pelo gestor central (servidor).



*Figura 10 - Estrutura do sistema (o número de gestores locais e de passageiros pode variar conforme a necessidade).*

As trocas de mensagens entre gestores locais e passageiros são realizadas do seguinte modo:

1. Passageiro envia *request* de mensagem para gestor central, contendo o texto da mensagem, a informação do passageiro que a está a enviar e a linha ferroviária afetada;
2. Gestor central reencaminha a mensagem para o gestor local;
3. Gestor local recebe a mensagem. De seguida tem a opção de enviar notificação de aviso a todos os passageiros referentes à linha ferroviária afetada. Se não o fizer, o fluxo acaba aqui. É enviado *request* ao gestor central com a notificação;
4. Gestor central reencaminha a notificação para todos os passageiros afetados.

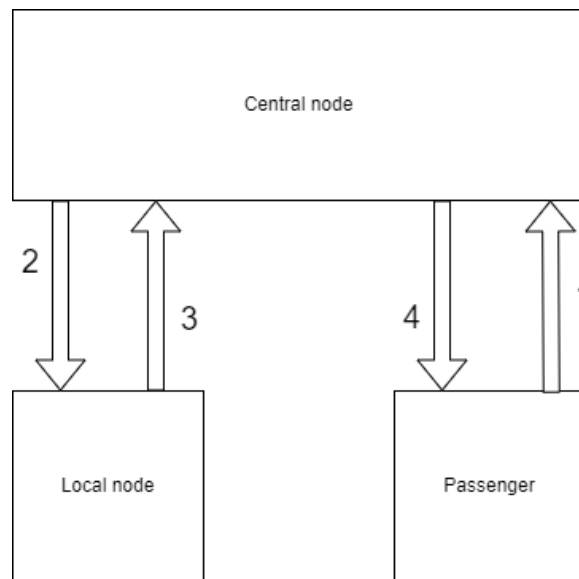


Figura 11 - Fluxo de lançamento de evento

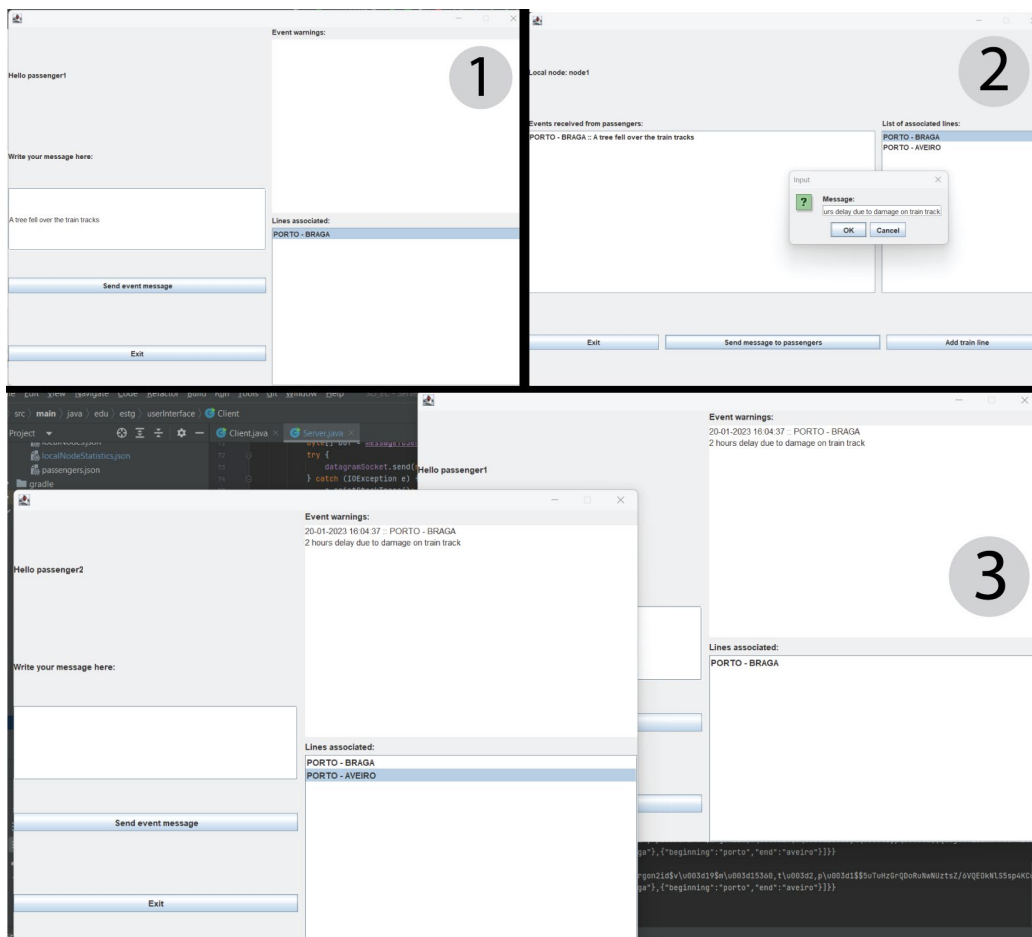


Figura 12 - Exemplo de evento lançado

Além disso, as informações relativas a passageiros, gestores locais e estatísticas são armazenadas em formato JSON na pasta /files.

Visto que os ficheiros que armazenam as informações relativas a gestores locais e passageiros também armazenam *passwords*, foi utilizado um método de *hashing* das *passwords* no ato do registo, de modo a estabelecer um nível de segurança das mesmas. Para o *hashing* das mesmas foi utilizado a função de derivação de chave *Argon2*<sup>[1]</sup>.

```
[
  {
    "name": "passenger2",
    "username": "passenger2",
    "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$QC6jvWT1EamM+Pfu+
AZ13ZNuH/QVB1DV2b54VXiaWLQ",
    "addedTrainLines": [
      {
        "beginning": "porto",
        "end": "braga"
      },
      {
        "beginning": "porto",
        "end": "aveiro"
      }
    ]
  },
  {
    "name": "passenger1",
    "username": "passenger1",
    "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$izhQYOYZkdm2DgXvt
XJ+Z1nnNAd2iKPKaiV0dbK7zd0",
    "addedTrainLines": [
      {
        "beginning": "porto",
        "end": "braga"
      }
    ]
  }
]
```

Figura 13 - Ficheiro *passengers.json*

```
[
  {
    "name": "node1",
    "username": "node1",
    "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$5uTuHzGrQDoRuNwNU
ztsZ/6VQEOkNlS5sp4KCUS3pHk",
    "passengers": [
      {
        "name": "passenger1",
        "username": "passenger1",
        "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$izhQYOYZkdm2DgXvt
XJ+ZlnnNAd2iKPKaiv0dbK7zd0",
        "addedTrainLines": [
          {
            "beginning": "porto",
            "end": "braga"
          }
        ]
      },
      {
        "name": "passenger2",
        "username": "passenger2",
        "password":
"$argon2id$v\u003d19$m\u003d15360,t\u003d2,p\u003d1$$QC6jvWT1EamM+Pfu+
AZl3ZNuH/QVB1DV2b54VXiaWLQ",
        "addedTrainLines": [
          {
            "beginning": "porto",
            "end": "braga"
          },
          {
            "beginning": "porto",
            "end": "aveiro"
          }
        ]
      }
    ]
  },
  {
    "trainLines": [
      {
        "beginning": "porto",
        "end": "braga"
      },
      {
        "beginning": "porto",
        "end": "aveiro"
      }
    ]
  }
]
]
```

Figura 14 - Ficheiro localNodes.json

No que toca às estatísticas, o gestor central (servidor) envia a cada 5 minutos um pedido de estatísticas a todos os gestores locais, que inclui o número de notificações lançadas, e o número de passageiros que receberam essas notificações. Ao receber, o gestor central soma essas estatísticas às já armazenadas e guarda num ficheiro local JSON de nome *localNodeStatistics.json*, para além de apresentar essas estatísticas no terminal.

```
[
  {
    "localNodeName": "node1",
    "messagesSent": 1,
    "passengersNotified": 2
  }
]
```

*Figura 15 - Ficheiro localNodeStatistics.json*

## Decisões tomadas

Ao longo do desenvolvimento do projeto, fez-se os possíveis para se tomar as melhores decisões, com base no que foi lecionado ao longo da Unidade Curricula de Sistemas Distribuídos e nos requerimentos para o sistema.

Entre estas decisões, está a da utilização de interfaces gráficas. Visto que este projeto faz uso de *multithreading*, observou-se que seria demasiado confuso o uso de terminal. Isto porque dada a presença de vários utilizadores em simultâneo, o manuseamento do sistema, bem como a visualização da informação, tornar-se-iam muito menos práticos.

Outra das decisões tomadas foi a apresentação dos eventos e notificações nas respetivas interfaces dos clientes em *plain-text*. Esta decisão foi tomada devido à falta de tempo na altura existente para a elaboração desta funcionalidade.

Uma terceira decisão foi a utilização de ficheiros JSON locais para armazenamento de dados. Optou-se por esta decisão, ao invés da utilização de uma base de dados, por questões de simplicidade e falta de tempo.



## Conclusão

O desenvolvimento deste projeto foi uma mais-valia na consolidação de conhecimentos acerca de *multithreading* e de ligações com recurso a *sockets*, lecionados na Unidade Curricular de Sistemas Distribuídos. Foi possível desenvolver um projeto interessante que simula sistemas reais, e que certamente foi uma excelente prática no uso de conhecimentos que serão úteis no futuro.

Foi possível implementar todas as funcionalidades pedidas no enunciado, ainda que com algumas ligeiras alterações, e o programa funciona como era esperado.

## Referências

- [1] – Argon2: <https://en.wikipedia.org/wiki/Argon2>, janeiro 2023.